



目 录

第 1 章 初识 Oracle.....	1
1.1 数据库基础	1
1.1.1 什么是数据库	1
1.1.2 表	2
1.1.3 数据类型	2
1.1.4 主键	2
1.2 数据库技术构成	3
1.2.1 数据库系统	3
1.2.2 SQL 语言	4
1.2.3 数据库访问技术	5
1.3 什么是 Oracle 12c.....	5
1.3.1 Oracle 的发展历程	6
1.3.2 Oracle 12c 版本的新功能.....	7
1.3.3 Oracle 的优势.....	8
1.4 Oracle 工具.....	9
1.4.1 SQL Plus.....	9
1.4.2 Oracle SQL Developer	10
1.5 如何学习 Oracle.....	11
第 2 章 Oracle 的安装与配置	12
2.1 安装 Oracle 12c.....	12
2.2 启动服务并登录 Oracle 数据库	18
2.2.1 启动 Oracle 服务	18
2.2.2 登录 Oracle 数据库	20
2.3 卸载 Oracle 12c.....	24
2.4 疑难解惑	27
2.5 经典习题	30



第 3 章 数据库和数据表的基本操作	31
3.1 创建数据库	31
3.2 删除数据库	34
3.3 创建数据表	38
3.3.1 创建数据表的语法形式	38
3.3.2 使用主键约束	39
3.3.3 使用外键约束	41
3.3.4 使用非空约束	43
3.3.5 使用唯一性约束	44
3.3.6 使用默认约束	45
3.3.7 使用检查约束	46
3.3.8 设置表的属性值自动增加	47
3.4 查看数据表结构	48
3.5 修改数据表	49
3.5.1 修改表名	49
3.5.2 修改字段的数据类型	50
3.5.3 修改字段名	51
3.5.4 添加字段	51
3.5.5 删除字段	52
3.6 删除数据表	53
3.6.1 删除没有被关联的表	53
3.6.2 删除被其他表关联的主表	54
3.7 综合案例——数据表的基本操作	55
3.8 疑难解惑	61
3.9 经典习题	63
第 4 章 数据类型和运算符	65
4.1 Oracle 数据类型介绍	65
4.1.1 数值类型	65
4.1.2 日期与时间类型	67
4.1.3 字符串类型	70
4.2 如何选择数据类型	71
4.3 常见运算符介绍	72
4.3.1 运算符概述	72
4.3.2 算术运算符	72
4.3.3 比较运算符	74
4.3.4 逻辑运算符	75



4.3.5	运算符的优先级	76
4.4	疑难解惑	77
4.5	经典习题	78
第 5 章	Oracle 函数	79
5.1	Oracle 函数简介	79
5.2	数学函数	79
5.2.1	绝对值函数 ABS(x)	80
5.2.2	平方根函数 SQRT(x) 和求余函数 MOD(x,y)	80
5.2.3	获取整数的函数 CEIL(x) 和 FLOOR(x)	80
5.2.4	获取随机数的函数 DBMS_RANDOM.RANDOM 和 DBMS_RANDOM.VALUE(x,y) ..	81
5.2.5	函数 ROUND(x)、ROUND(x,y) 和 TRUNC(x,y)	81
5.2.6	符号函数 SIGN(x)	83
5.2.7	幂运算函数 POWER(x,y) 和 EXP(x)	83
5.2.8	对数运算函数 LOG(x,y) 和 LN(x)	83
5.2.9	正弦函数 SIN(x) 和反正弦函数 ASIN(x)	84
5.2.10	余弦函数 COS(x) 和反余弦函数 ACOS(x)	84
5.2.11	正切函数 TAN(x) 和反正切函数 ATAN(x)	85
5.3	字符串函数	85
5.3.1	计算字符串长度的函数	85
5.3.2	合并字符串函数 CONCAT(s1,s2)	86
5.3.3	字符串搜索函数 INSTR(s,x)	86
5.3.4	字母大小写转换函数	86
5.3.5	获取指定长度的字符串的函数 SUBSTR(s,m,n)	87
5.3.6	替换字符串的函数 REPLACE(s1,s2,s3)	87
5.3.7	删除字符串首尾指定字符的函数 LTRIM(s,n) 和 RTRIM(s,n)	88
5.3.8	删除指定字符串的函数 TRIM()	88
5.3.9	字符集名称和 ID 互换函数	89
5.4	日期和时间函数	89
5.4.1	获取当前日期和时间的函数	89
5.4.2	获取时区的函数	90
5.4.3	获取指定月份最后一天的函数	90
5.4.4	获取指定日期后一周的日期的函数	91
5.4.5	获取指定日期特定部分的函数	91
5.4.6	获取两个日期之间的月份数的函数	91
5.5	转换函数	92
5.5.1	字符串转 ASCII 类型字符串函数	92
5.5.2	二进制转十进制函数	92



5.5.3	数据类型转换函数	92
5.5.4	数值转换为字符串函数	93
5.5.5	字符转日期函数	93
5.5.6	字符串转数字函数	94
5.6	系统信息函数	94
5.6.1	返回登录名函数	94
5.6.2	返回会话以及上下文信息函数	94
5.7	综合案例——Oracle 函数的使用	95
5.8	疑难解惑	97
5.9	经典习题	98
第 6 章	查询数据	99
6.1	基本查询语句	99
6.2	单表查询	102
6.2.1	查询所有字段	102
6.2.2	查询指定字段	103
6.2.3	查询指定记录	105
6.2.4	带 IN 关键字的查询	107
6.2.5	带 BETWEEN AND 的范围查询	108
6.2.6	带 LIKE 的字符匹配查询	110
6.2.7	查询空值	111
6.2.8	带 AND 的多条件查询	113
6.2.9	带 OR 的多条件查询	114
6.2.10	查询结果不重复	115
6.2.11	对查询结果排序	117
6.2.12	分组查询	121
6.2.13	使用 ROWNUM 限制查询结果的数量	127
6.3	使用聚合函数查询	127
6.3.1	COUNT()函数	128
6.3.2	SUM()函数	129
6.3.3	AVG()函数	130
6.3.4	MAX()函数	130
6.3.5	MIN()函数	132
6.4	连接查询	133
6.4.1	内连接查询	133
6.4.2	外连接查询	137
6.4.3	复合条件连接查询	139
6.5	子查询	140



6.5.1	带 ANY、SOME 关键字的子查询	140
6.5.2	带 ALL 关键字的子查询	141
6.5.3	带 EXISTS 关键字的子查询	141
6.5.4	带 IN 关键字的子查询	143
6.5.5	带比较运算符的子查询	145
6.6	合并查询结果	146
6.7	为表和字段取别名	150
6.7.1	为表取别名	150
6.7.2	为字段取别名	151
6.8	使用正则表达式查询	153
6.8.1	查询以特定字符或字符串开头的记录	154
6.8.2	查询以特定字符或字符串结尾的记录	155
6.8.3	用符号"."来替代字符串中的任意一个字符	155
6.8.4	使用"*"和"+"来匹配多个字符	156
6.8.5	匹配指定字符串	156
6.8.6	匹配指定字符中的任意一个	158
6.8.7	匹配指定字符以外的字符	159
6.8.8	使用{n,}或者{n,m}来指定字符串连续出现的次数	159
6.9	综合案例——数据表查询操作	160
6.10	疑难解惑	168
6.11	经典习题	168
第 7 章	插入、更新与删除数据	170
7.1	插入数据	170
7.1.1	为表的所有字段插入数据	170
7.1.2	为表的指定字段插入数据	172
7.1.3	同时插入多条记录	173
7.1.4	将查询结果插入到表中	175
7.2	更新数据	177
7.3	删除数据	179
7.4	综合案例——记录的插入、更新和删除	181
7.5	疑难解惑	186
7.6	经典习题	187
第 8 章	视图	188
8.1	视图概述	188
8.1.1	视图的含义	188
8.1.2	视图的作用	189

8.2	创建视图	190
8.2.1	创建视图的语法形式	190
8.2.2	在单表上创建视图	190
8.2.3	在多表上创建视图	191
8.2.4	创建视图的视图	192
8.2.5	创建没有源表的视图	193
8.3	查看视图	193
8.4	修改视图	194
8.4.1	CREATE OR REPLACE VIEW 语句修改视图	194
8.4.2	ALTER 语句修改视图的约束	195
8.5	更新视图	196
8.6	删除视图	198
8.7	限制视图的数据操作	198
8.7.1	设置视图的只读属性	199
8.7.2	设置视图的检查属性	199
8.8	综合案例——视图应用	199
8.9	疑难解惑	205
8.10	经典习题	206
第 9 章	游标	207
9.1	认识游标	207
9.1.1	游标的概念	207
9.1.2	游标的优点	207
9.1.3	游标的分类	208
9.2	显式游标	208
9.2.1	显式游标的语法	208
9.2.2	打开游标	209
9.2.3	读取游标中的数据	209
9.2.4	关闭游标	209
9.2.5	使用显式游标的案例	209
9.2.6	使用显式游标的 LOOP 语句	210
9.2.7	使用 BULK COLLECT 和 FOR 语句的游标	211
9.2.8	使用 CURSOR FOR LOOP 语句的游标	212
9.2.9	显式游标的属性	213
9.3	隐式游标	217
9.3.1	使用隐式游标	217
9.3.2	隐式游标的属性	218
9.3.3	在游标中使用异常处理	220

9.4 综合案例——游标的综合应用	221
9.5 疑难解惑	223
9.6 经典习题	223
第 10 章 存储过程	224
10.1 创建存储过程	224
10.1.1 什么是存储过程	224
10.1.2 创建存储过程	225
10.2 调用存储过程	226
10.3 查看存储过程	227
10.4 存储过程的参数	227
10.4.1 无参数的存储过程	227
10.4.2 有参数的存储过程	229
10.5 修改存储过程	230
10.6 删除存储过程	231
10.7 查看存储过程的错误	231
10.8 综合案例——综合运用存储过程	232
10.9 疑难解惑	234
10.10 经典习题	234
第 11 章 Oracle 触发器	235
11.1 创建触发器	235
11.1.1 什么是触发器	235
11.1.2 创建只有一个执行语句的触发器	236
11.1.3 创建有多个执行语句的触发器	236
11.2 查看触发器	239
11.2.1 查看触发器的名称	239
11.2.2 查看触发器的内容信息	240
11.3 触发器的使用	240
11.4 修改触发器	242
11.5 删除触发器	242
11.6 综合案例——使用触发器	243
11.7 疑难解惑	245
11.8 经典习题	245
第 12 章 管理表空间	246
12.1 什么是表空间	246
12.2 查看表空间	247

12.3	管理表空间	248
12.3.1	创建表空间	248
12.3.2	设置表空间的可用状态	250
12.3.3	设置表空间的读写状态	250
12.3.4	重命名表空间	251
12.3.5	删除表空间	251
12.3.6	建立大文件表空间	251
12.4	管理临时表空间	252
12.4.1	创建临时表空间	252
12.4.2	查看临时表空间	252
12.4.3	创建临时表空间组	253
12.4.4	查看临时表空间组	253
12.4.5	删除临时表空间组	254
12.5	管理数据文件	254
12.5.1	移动数据文件	254
12.5.2	删除数据文件	254
12.6	疑难解惑	255
12.7	经典习题	255
第 13 章	事务与锁	256
13.1	事务管理	256
13.1.1	什么是事务	256
13.1.2	事务的属性	257
13.1.3	事务管理的常用语句	257
13.1.4	事务的类型	257
13.1.5	事务的应用实例	258
13.1.6	事务的保存点	259
13.2	锁	260
13.2.1	什么是锁	261
13.2.2	锁的分类	262
13.2.3	锁的类型	262
13.2.4	锁等待和死锁	263
13.3	综合案例——死锁的案例	265
13.4	疑难解惑	265
13.5	经典习题	266
第 14 章	Oracle 的用户管理	267
14.1	账户管理	267

14.1.1	管理账号概述	267
14.1.2	新建普通用户	268
14.1.3	修改用户信息	269
14.1.4	删除用户	270
14.2	权限管理	270
14.2.1	授权	270
14.2.2	收回权限	271
14.2.3	查看权限	272
14.3	角色管理	273
14.3.1	角色概述	273
14.3.2	创建角色	273
14.3.3	设置角色	274
14.3.4	修改角色	275
14.3.5	查看角色	275
14.3.6	删除角色	275
14.4	管理概要文件 PROFILE	276
14.4.1	PROFILE 概述	276
14.4.2	创建概要文件	276
14.4.3	修改概要文件	277
14.4.4	删除概要文件	277
14.5	疑难解惑	277
14.6	经典习题	278
第 15 章	控制文件和日志	279
15.1	控制文件简介	279
15.2	控制文件的应用案例	280
15.2.1	查看控制文件的内容	280
15.2.2	更新控制文件的内容	280
15.2.3	使用 init.ora 多路复用控制文件	281
15.2.4	使用 SPFILE 多路复用控制文件	282
15.2.5	创建控制文件	282
15.3	日志简介	285
15.4	管理日志文件	286
15.4.1	新建日志文件组	286
15.4.2	添加日志文件到日志文件组	287
15.4.3	删除日志文件组和日志文件	287
15.4.4	查询日志文件组和日志文件	288
15.5	疑难解惑	289

15.6 经典习题	289
第 16 章 数据备份与还原	290
16.1 数据备份	290
16.1.1 冷备份	290
16.1.2 热备份	291
16.2 数据还原	292
16.3 表的导出和导入	293
16.3.1 用 EXP 工具导出数据	294
16.3.2 用 EXPDP 导出数据	294
16.3.3 用 IMP 导入数据	295
16.3.4 用 IMPDP 导入数据	296
16.4 疑难解惑	296
16.5 经典习题	297
第 17 章 性能优化	298
17.1 优化简介	298
17.1.1 修改系统全局区	298
17.1.2 修改进程全局区	300
17.2 优化查询	301
17.2.1 分析查询语句的执行计划	301
17.2.2 索引对查询速度的影响	304
17.2.3 使用索引查询	304
17.2.4 优化子查询	305
17.3 优化数据库结构	305
17.3.1 将字段很多的表分解成多个表	305
17.3.2 增加中间表	307
17.3.3 增加冗余字段	308
17.3.4 优化插入记录的速度	309
17.4 优化 Oracle 服务器	310
17.4.1 优化服务器硬件	310
17.4.2 优化 Oracle 的参数	311
17.5 疑难解惑	313
17.6 经典习题	313
第 18 章 设计新闻发布系统数据库	314
18.1 系统概述	314
18.2 系统功能	315

18.3 数据库设计和实现	316
18.3.1 设计表	316
18.3.2 设计索引	321
18.3.3 设计视图	322
18.3.4 设计触发器	322
18.4 小结	323
第 19 章 设计论坛管理系统数据库	324
19.1 系统概述	324
19.2 系统功能	325
19.3 数据库设计和实现	326
19.3.1 设计方案图表	326
19.3.2 设计表	328
19.3.3 设计索引	332
19.3.4 设计视图	333
19.3.5 设计触发器	333
19.4 小结	335

第 1 章

◀ 初识Oracle ▶

Oracle 是以关系数据库为数据存储和管理作为构架基础, 构建出的数据库管理系统。Oracle 是世界上第一个支持 SQL 语言的商业数据库, 定位于高端工作站, 以及作为服务器的小型计算机, 如 IBM P 系列服务器、HP 的 Integrity 服务器和 Sun Fire 服务器等。本章主要介绍数据库的基础知识, 通过本章的学习, 读者可以了解数据库的基本概念、数据库的构成和 Oracle 的基本知识。

- 了解什么是数据库
- 掌握什么是表、数据类型和主键
- 熟悉数据库的技术构成
- 熟悉什么是 Oracle
- 掌握常见的 Oracle 工具
- 了解如何学习 Oracle

1.1 数据库基础

数据库由一批数据构成有序的集合, 这些数据被存放在结构化的数据表里。数据表之间相互关联, 反映了客观事物间的本质联系。数据库系统提供对数据的安全控制和完整性控制。本节将介绍数据库中的一些基本概念, 包括: 数据库的定义、数据表的定义和数据类型等。

1.1.1 什么是数据库

数据库的概念诞生于 60 年前, 随着信息技术和市场的快速发展, 数据库技术层出不穷, 随着应用的拓展和深入, 数据库的数量和规模越来越大, 其诞生和发展给计算机信息管理带来了一场巨大的革命。

数据库的发展大致划分为如下几个阶段: 人工管理阶段、文件系统阶段、数据库系统阶段、高级数据库阶段。其种类大概有 3 种: 层次式数据库、网络式数据库和关系式数据库。不同种类的数据库按不同的数据结构来联系和组织。

对于数据库的概念, 没有一个完全固定的定义, 随着数据库历史的发展, 定义的内容也有

很大的差异，其中一种比较普遍的观点认为，数据库（DataBase，DB）是一个长期存储在计算机内的、有组织的、有共享的、统一管理的数据集合。它是一个按数据结构来存储和管理数据的计算机软件系统。即数据库包含两层含义：保管数据的“仓库”，以及数据管理的方法和技术。

数据库的特点包括：实现数据共享，减少数据冗余；采用特定的数据类型；具有较高的数据独立性；具有统一的数据控制功能。

1.1.2 表

在关系数据库中，数据表是一系列二维数组的集合，用来存储数据和操作数据的逻辑结构。它由纵向的列和横向的行组成，行被称为记录，是组织数据的单位；列被称为字段，每一列表示记录的一个属性，都有相应的描述信息，如数据类型、数据宽度等。

例如，一个有关作者信息的名为 authors 的表中，每个列包含所有作者的某个特定类型的信息，比如“姓名”，而每行则包含了某个特定作者的所有信息：编号、姓名、性别、专业，如图 1-1 所示。

编号	姓名	性别	专业
100	张三	f	计算机
101	李芬	m	会计
102	岳阳	f	园林

图 1-1 authors 表结构与记录

1.1.3 数据类型

数据类型决定了数据在计算机中的存储格式，代表不同的信息类型。常用的数据类型有：整数数据类型、浮点数数据类型、精确小数类型、二进制数据类型、日期/时间数据类型、字符串数据类型。

表中的每一个字段就是某种指定数据类型，比如图 1-1 中“编号”字段为整数型数据，“性别”字段为字符型数据。

1.1.4 主键

主键（PRIMARY KEY）又称主码，用于唯一地标识表中的每一条记录。可以定义表中的一列或多列为主键，主键列上不能有两行相同的值，也不能为空值。假如定义 authors 表，该表给每一个作者分配一个“作者编号”，该编号作为数据表的主键，如果出现相同的值，将提示错误，系统不能确定查询的究竟是哪一条记录；如果把作者的“姓名”作为主键，则不能出现重复的名字，这与现实不相符合，因此“姓名”字段不适合作为主键。

1.2 数据库技术构成

数据库系统由硬件部分和软件部分共同构成,硬件部分主要用于存储数据库中的数据,包括计算机、存储设备等;软件部分则主要包括 DBMS、支持 DBMS 运行的操作系统,以及支持多种语言进行应用开发的访问技术等。本节将介绍数据库的技术构成。

1.2.1 数据库系统

数据库系统有 3 个主要的组成部分。

- 数据库: 用于存储数据的地方。
- 数据库管理系统: 用于管理数据库的软件。
- 数据库应用程序: 为了提高数据库系统的处理能力所使用的管理数据库的软件补充。

数据库 (DataBase) 提供了一个存储空间用以存储各种数据,可以将数据库视为一个存储数据的容器。一个数据库可能包含许多文件,一个数据库系统中通常包含许多数据库。

数据库管理系统 (DataBase Management System, DBMS) 是用户创建、管理和维护数据库时所使用的软件,位于用户与操作系统之间,对数据库进行统一管理。DBMS 能定义数据存储结构,提供数据的操作机制,维护数据库的安全性、完整性和可靠性。

数据库应用程序 (DataBase Application), 虽然已经有了 DBMS, 但是在很多情况下, DBMS 无法满足对数据管理的要求。数据库应用程序的使用可以满足对数据管理的更高要求, 还可以使数据管理过程更加直观和友好。数据库应用程序负责与 DBMS 进行通信, 访问和管理 DBMS 中存储的数据, 允许用户插入、修改、删除 DB 中的数据。

数据库系统如图 1-2 所示。

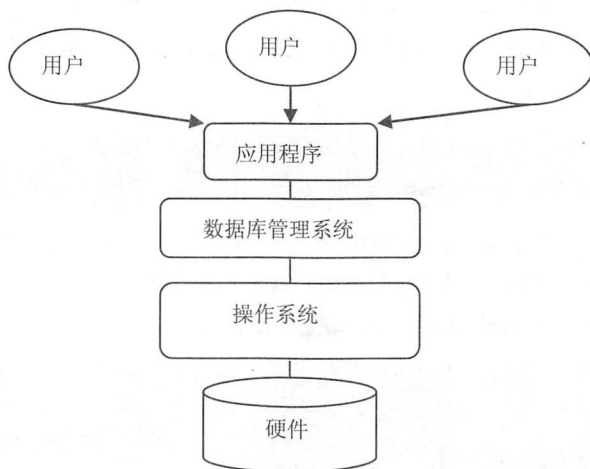


图 1-2 数据库系统

1.2.2 SQL 语言

对数据库进行查询和修改操作的语言叫做 SQL。SQL 的含义是结构化查询语言 (Structured Query Language)。SQL 有许多不同的类型，有 3 个主要的标准：ANSI（美国国家标准机构）SQL，对 ANSI SQL 修改后在 1992 年采纳的标准，称为 SQL-92 或 SQL2。最近的 SQL-99 标准，SQL-99 标准从 SQL2 扩充而来，并增加了对对象关系特征和许多其他新功能。其次，各大数据库厂商提供不同版本的 SQL，这些版本的 SQL 不但能包括原始的 ANSI 标准，而且在很大程度上支持新推出的 SQL-92 标准。

SQL 语言包含以下 4 个部分。

- (1) 数据定义语言 (DDL): DROP、CREATE、ALTER 等语句。
- (2) 数据操作语言 (DML): INSERT (插入)、UPDATE (修改)、DELETE (删除) 语句。
- (3) 数据查询语言 (DQL): SELECT 语句。
- (4) 数据控制语言 (DCL): GRANT、REVOKE、COMMIT、ROLLBACK 等语句。

下面是一条 SQL 语句的例子，该语句声明创建一个叫 students 的表：

```
CREATE TABLE students
(
  student_id number(11),
  name VARCHAR2(30),
  sex CHAR(2),
  PRIMARY KEY (student_id)
);
```

该表包含 3 个字段，分别为 student_id、name、sex，其中 student_id 定义为表的主键。

现在只是定义了一张表格，但并没有任何数据，接下来这条 SQL 声明语句，将在 students 表中插入一条数据记录：

```
INSERT INTO students (student_id, name, sex, birth)
VALUES (41048101, 'Lucy Green', '1');
```

执行完该 SQL 语句之后，students 表中就会增加一行新记录，该记录中字段 student_id 的值为 41048101，name 字段的值为 Lucy Green，sex 字段的值为 1。

再使用 SELECT 查询语句获取刚才插入的数据，如下：

```
SELECT name FROM students WHERE student_id = 41048101;

NAME
-----
Lucy Green
```

上面简单列举了常用的数据库操作语句，在这里给读者一个直观的印象，读者可能还不能

理解，接下来会在学习 Oracle 的过程中详细介绍这些知识。

1.2.3 数据库访问技术

不同的程序设计语言会有各自不同的数据库访问技术，程序语言通过这些技术，执行 SQL 语句，进行数据库管理。主要的数据库访问技术有：ODBC、JDBC、ADO.NET 和 PDO。

1. ODBC

Open DataBase Connectivity（开放数据库互连）技术为访问不同的 SQL 数据库提供了一个共同的接口。ODBC 使用 SQL 作为访问数据的标准。这一接口提供了最大限度的互操作性：一个应用程序可以通过共同的一组代码访问不同的 SQL 数据库管理系统（DBMS）。

一个基于 ODBC 的应用程序对数据库的操作不依赖任何 DBMS，不直接与 DBMS 打交道，所有的数据库操作由对应的 DBMS 的 ODBC 驱动程序完成。也就是说，不论是 Access、Oracle 还是 Oracle 数据库，均可用 ODBC API 进行访问。由此可见，ODBC 的最大优点是能以统一的方式处理所有的数据库。

2. JDBC

Java DataBase Connectivity（Java 数据库连接）用于 Java 应用程序连接数据库的标准方法，是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成。

3. ADO.NET

ADO.NET 是微软在 .NET 框架下开发设计的一组用于和数据源进行交互的面向对象类库。ADO.NET 提供了对关系数据、XML 和应用程序数据的访问，允许和不同类型的数据源以及数据库进行交互。

4. PDO

PDO（PHP Data Object）为 PHP 访问数据库定义了一个轻量级的、一致性的接口，它提供了一个数据访问抽象层，这样无论使用什么数据库，都可以通过一致的函数执行查询和获取数据。PDO 是 PHP 5 新加入的一个重大功能。

1.3 什么是 Oracle 12c

Oracle 数据库是积聚了众多领先性的数据库系统，在集群技术、高可用性、商业智能、安全性、系统管理等方面都领跑业界。Oracle 是一个大型关系数据库管理系统，目前已经成为企业级开发的首选。本章节主要介绍 Oracle 数据库的发展历史和 Oracle 12c 的新功能。

1.3.1 Oracle 的发展历程

Oracle 是由甲骨文公司开发的，并于 1989 年正式进入中国市场，成为第一家进入中国的世界软件巨头。Oracle 大致发展历程如下：

1977 年，Larry Ellison、Bob Miner 和 Ed Oates 等人组建了 Relational 软件公司（Relational Software Inc., RSI）。他们决定使用 C 语言和 SQL 界面构建一个关系数据库管理系统（Relational DataBase Management System, RDBMS），并很快发布了第一个版本（仅是原型系统）。

1979 年，RSI 首次向客户发布了产品，即第 2 版。该版本的 RDBMS 可以在装有 RSX-11 操作系统的 PDP-11 机器上运行，后来又移植到了 DEC VAX 系统。

1983 年，发布的第 3 个版本中加入了 SQL 语言，而且性能也有所提升，其他功能也得到增强。与前几个版本不同的是，这个版本是完全用 C 语言编写的。同年，RSI 更名为 Oracle Corporation，也就是今天的 Oracle 公司。

1984 年，Oracle 的第 4 版发布。该版本既支持 VAX 系统，也支持 IBM VM 操作系统。这也是第一个加入了读一致性（Read-consistency）的版本。

1985 年，Oracle 的第 5 版发布。该版本可称作是 Oracle 发展史上的里程碑，因为它通过 SQL*Net 引入了客户端/服务器的计算机模式，同时它也是第一个打破 640KB 内存限制的 MS-DOS 产品。

1988 年，Oracle 的第 6 版发布。该版本除了改进性能、增强序列生成与延迟写入（Deferred Writes）功能以外，还引入了底层锁。除此之外，该版本还加入了 PL/SQL 和热备份等功能。这时 Oracle 已经可以在许多平台和操作系统上运行。

1991 年，Oracle RDBMS 的 6.1 版在 DEC VAX 平台中引入了 Parallel Server 选项，很快该选项也可用于许多其他平台。

1992 年，Oracle 7 发布。Oracle 7 在对内存、CPU 和 I/O 的利用方面作了许多体系结构上的变动，这是一个功能完整的关系数据库管理系统，在易用性方面也作了许多改进，引入了 SQL*DBA 工具和 database 角色。

1997 年，Oracle 8 发布。Oracle 8 除了增加许多新特性和管理工具以外，还加入了对象扩展（Object Extension）特性。Oracle 开始在 Windows 系统下使用，以前的版本都是在 UNIX 环境下运行。

2001 年，Oracle 9i release 1 发布。这是 Oracle 9i 的第一个发行版，包含 RAC（Real Application Cluster）等新功能。

2002 年，Oracle 9i release 2 发布，它在 release 1 的基础上增加了集群文件系统（Cluster File System）等特性。

2004 年，针对网格计算的 Oracle 10g 发布。该版本中 Oracle 的功能、稳定性和性能的实现都达到了一个新的水平。

2007 年 7 月 12 日，甲骨文公司推出最新数据库软件 Oracle 11g，Oracle 11g 有 400 多项功能，经过了 1500 万个小时的测试，开发工作量达到了 3.6 万人/月。相对以往版本而言，Oracle 11g 具有了与众不同的特性。

2013年6月26日, Oracle Database 12c 版本正式发布, 12c 中的 c 是 cloud, 代表云计算的意思。

与 Oracle 数据库基本同时期的还有 Informix 数据库系统。两者使用的用户有所侧重。Oracle 数据库系统银行业使用较多, Informix 数据库系统通信业使用较多。由于 Oracle 数据库产品是当前数据库技术的典型代表, 除了数据库系统外, 还有应用系统和开发工具等。

1.3.2 Oracle 12c 版本的新功能

新版 Oracle Database 12c 汇集了参会者最多的目光, Larry Ellison 也在开幕演讲中重点介绍了 12c 的一些新特性。在学习 Oracle Database 12c 之前, 数据库管理员希望能够提前了解它的一些新功能、新特性。

(1) PL/SQL 性能增强: 类似在匿名块中定义过程, 现在可以通过 WITH 语句在 SQL 中定义一个函数, 采用这种方式可以提高 SQL 调用的性能。

(2) 改善 Defaults: 包括序列作为默认值; 自增列; 当明确插入 NULL 时指定默认值; metadata-only default 值指的是增加一个新列时指定的默认值, 和 11g 中的区别在于, 11g 的 default 值要求 NOT NULL 列。

(3) 放宽多种数据类型长度限制: 增加了 VARCHAR2、NVARCHAR2 和 RAW 类型的长度到 32KB, 要求兼容性设置为 12.0.0.0 以上, 且设置了初始化参数 MAX_SQL_STRING_SIZE 为 EXTENDED, 这个功能不支持 CLUSTER 表和索引组织表; 最后这个功能并不是真正改变了 VARCHAR2 的限制, 而是通过 OUT OF LINE 的 CLOB 实现。

(4) TOP N 的语句实现: 在 SELECT 语句中使用“FETCH next N rows”或者“OFFSET”, 可以指定前 N 条或前百分之多少的记录。

(5) 行模式匹配: 类似分析函数的功能, 可以在行间进行匹配判断并进行计算。在 SQL 中新的模式匹配语句是“match_recognize”。

(6) 分区改进: Oracle Database 12c 中对分区功能做了较多的调整, 其中共分成 6 个部分。

- INTERVAL-REFERENCE 分区: 把 11g 的 INTERVAL 分区和 REFERENCE 分区结合, 这样主表自动增加一个分区后, 所有的子表、孙子表、重孙子表、重重重...孙子表都可以自动随着外接列新数据增加, 自动创建新的分区。
- TRUNCATE 和 EXCHANGE 分区及子分区: 无论是 TRUNCATE 还是 EXCHANGE 分区, 在主表上执行, 都可以级联的作用在子表、孙子表、重孙子表、重重重...孙子表上同时运行。对于 TRUNCATE 而言, 所有表的 TRUNCATE 操作在同一个事务中, 如果中途失败, 会回滚到命令执行之前的状态。这两个功能通过关键字 CASCADE 实现。
- 在线移动分区: 通过 MOVE ONLINE 关键字实现在线分区移动。在移动的过程中, 对表和被移动的分区可以执行查询、DML 语句以及分区的创建和维护操作。整个移动过程对应用透明。这个功能极大地提高了整体可用性, 缩短了分区维护窗口。
- 多个分区同时操作: 可以对多个分区同时进行维护操作, 比如将一年的 12 个分区 MERGE 到 1 个新的分区中, 或将一个分区 SPLIT 成多个分区。可以通过 FOR 语句指

定操作的每个分区，对于 RANGE 分区而言，也可以通过 TO 来指定处理分区的范围。多个分区同时操作自动并行完成。

- 异步全局索引维护：对于非常大的分区表而言，UPDATE GLOBAL INDEX 不再是痛苦。Oracle 可以实现异步全局索引维护的功能，即使是几亿条记录的全局索引，在分区维护操作，比如 DROP 或 TRUNCATE 后，仍然是 VALID 状态，索引不会失效，不过索引的状态是包含 OBSOLETE 数据，当维护操作完成，索引状态恢复。
- 部分本地和全局索引：Oracle 的索引可以在分区级别定义。无论全局索引还是本地索引都可以在分区表的部分分区上建立，其他分区上则没有索引。当通过索引列访问全表数据时，Oracle 通过 UNION ALL 实现，一部分通过索引扫描，另一部分通过全分区扫描。这可以减少对历史数据的索引量，极大地增加了灵活性。

(7) Adaptive 执行计划：拥有学习功能的执行计划，Oracle 会把实际运行过程中读取到的返回结果作为进一步执行计划判断的输入，因此统计信息不准确或查询真正结果与计算结果不准时，可以得到更好的执行计划。

(8) 统计信息增强：动态统计信息收集增加第 11 层，使得动态统计信息收集的功能更强；增加了混合统计信息用以支持包含大量不同值，且个别值数据倾斜的情况；添加了数据加载过程收集统计信息的能力；对于临时表增加了会话私有统计信息。

(9) 临时 UNDO：将临时段的 UNDO 独立出来，放到 TEMP 表空间中，优点包括：减少 UNDO 产生的数量；减少 REDO 产生的数量；在 ACTIVE DATA GUARD 上允许对临时表进行 DML 操作。

(10) 数据优化：新增了 ILM（数据生命周期管理）功能，添加了“数据库热图”（Database Heat Map），在视图中直接看到数据的利用率，找到哪些数据是最“热”的数据。可以实现数据的在线压缩和数据分级，其中数据分级可以在线将定义时间内的数据文件转移到归档存储，也可以将数据表定时转移至归档文件。另外，也可以实现在线的数据压缩。

(11) 应用连续性：Oracle Database 12c 之前 RAC 的 FAILOVER 只做到 SESSION 和 SELECT 级别，对于 DML 操作无能为力，当设置为 SESSION 时，进行到一半的 DML 自动回滚；而对于 SELECT，虽然 FAILOVER 可以不中断查询，但是对于 DML 的问题更甚之，必须要手工回滚。而 Oracle Database 12c 中，Oracle 终于支持事务的 FAILOVER。

(12) Oracle Pluggable Database：Oracle PDB 体系结构由一个容器数据库（CDB）和多个可组装式数据库（PDB）构成，PDB 包含独立的系统表空间和 SYSAUX 表空间等，但是所有 PDB 共享 CDB 的控制文件、日志文件和 UNDO 表空间。

1.3.3 Oracle 的优势

Oracle 的主要优势如下。

- (1) 速度：运行速度快。
- (2) 稳定性：Oracle 是目前数据库中稳定性非常好的数据库。
- (3) 共享 SQL 和多线索服务器体系结构：自 Oracle 7.x 以来引入了共享 SQL 和多线索服务器体系结构。这减少了 Oracle 的资源占用，并增强了 Oracle 的能力，使之在低档软硬件平

台上用较少的资源就可以支持更多的用户，而在高档平台上可以支持成百上千个用户。

(4) 可移植性：能够工作在不同的系统平台上，例如 Windows 和 Linux 等。

(5) 安全性强：提供了基于角色 (Role) 分工的安全保密管理，在数据库管理功能、完整性检查、安全性、一致性方面都有良好的表现。

(6) 支持类型多：支持大量多媒体数据，如二进制图形、声音、动画以及多维数据结构等。

(7) 方便管理数据：提供了新的分布式数据库能力，可通过网络较方便地读写远端数据库里的数据，并有对称复制的技术。

1.4 Oracle 工具

Oracle 数据库管理系统提供了许多命令行工具，这些工具可以用来管理 Oracle 服务器、对数据库进行访问控制、管理 Oracle 用户以及数据库备份和恢复工具等。而且，Oracle 提供图形化的管理工具，这使得对数据库的操作更加简单。本节将为读者介绍这些工具的作用。

1.4.1 SQL Plus

SQL Plus 为客户端工具。在 SQL Plus 中，可以运行 SQL Plus 命令与 SQL 语句。

通常所说的 DML、DDL、DCL 语句都是 SQL 语句，它们执行完后，都可以保存在一个被称为 SQL Buffer 的内存区域中，并且只能保存一条最近执行的 SQL 语句，可以对保存在 SQL Buffer 中的 SQL 语句进行修改，然后再次执行，SQL Plus 一般都与数据库打交道。

除了 SQL 语句，在 SQL Plus 中执行的其他语句称之为 SQL Plus 命令。它们执行完后，不保存在 SQL Buffer 的内存区域中，它们一般用来对输出的结果进行格式化显示，以便于制作报表。

SQL Plus 是目前最常用的工具，具有很强的功能，主要功能包含如下：

- (1) 数据库的维护，如启动、关闭等，这一般在服务器上操作。
- (2) 执行 SQL 语句。
- (3) 执行 SQL 脚本。
- (4) 数据导出为报表。
- (5) 应用程序开发、测试 SQL。
- (6) 生成新的 SQL 脚本。
- (7) 供应用程序调用，如安装程序中进行脚本的安装。
- (8) 用户管理及权限维护等。

SQL Plus 的运行界面如图 1-3 所示。

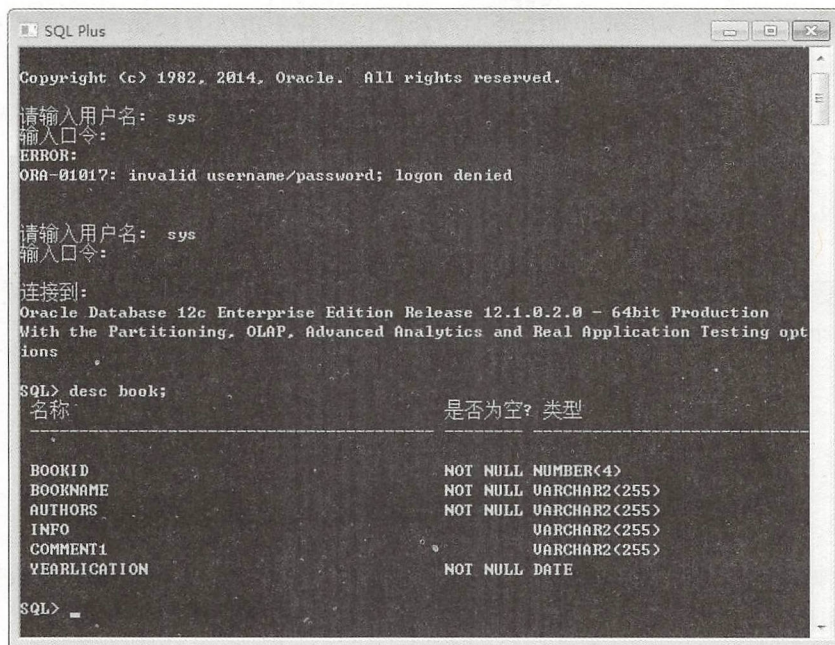


图 1-3 SQL Plus 运行界面

1.4.2 Oracle SQL Developer

Oracle SQL Developer 是 Oracle 公司出品的一个免费的集成开发环境。使用 SQL Developer 可以浏览数据库对象、运行 SQL 语句和脚本、编辑和调试 PL/SQL 语句。另外，还可以创建执行和保存报表。Oracle SQL Developer 可以连接任何 Oracle 9.2.0.1 或者以上版本的 Oracle 数据库，支持 Windows、Linux 和 Mac OS X 系统。

Oracle 12c 系统自带 SQL Developer 开发工具，操作主界面如图 1-4 所示。

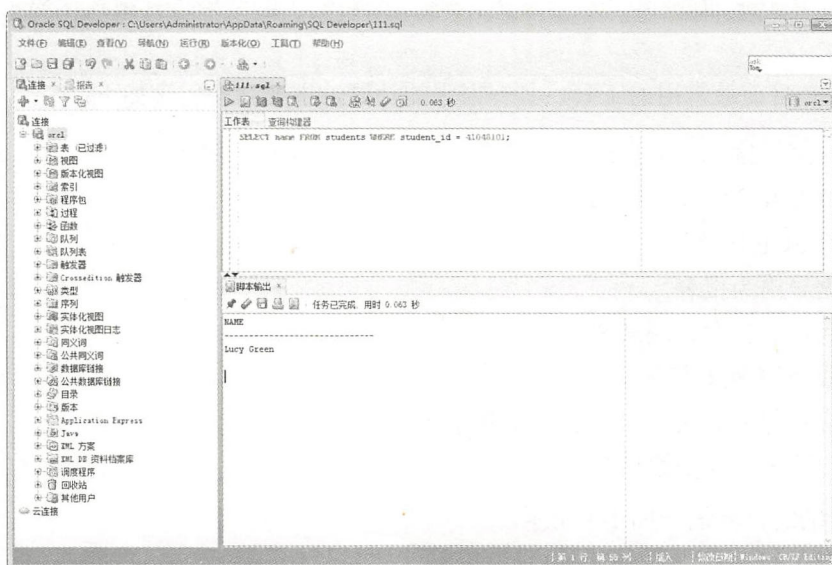


图 1-4 SQL Developer 主界面

1.5 如何学习 Oracle

在学习 Oracle 数据库之前，很多读者都会问如何才能学习好 Oracle 的相关技能。下面就来讲述学习 Oracle 的方法。

1. 培养兴趣

兴趣是最好的老师，不论学习什么知识，兴趣都可以极大地提高学习效率。当然，学习 Oracle 也不例外。

2. 夯实基础

计算机领域的技术非常强调基础，刚开始学习可能还认识不到这一点，随着技术应用的深入，只有具备扎实的基础功底，才能在技术的道路上走得更快、更远。对于 Oracle 的学习来说，SQL 语句是其中最为基础的部分，很多操作都是通过 SQL 语句来实现的。所以在学习的过程中，读者要多编写 SQL 语句，对于同一个功能，使用不同的实现语句来完成，从而深刻理解其不同之处。

3. 及时学习新知识

正确、有效地利用搜索引擎，可以搜索到很多关于 Oracle 12c 的相关知识。同时，参考别人解决问题的思路，也可以吸取别人的经验，及时获取最新的技术资料。

4. 多实践操作

数据库系统具有极强的操作性，需要多动手上机操作。在实际操作的过程中才能发现问题，并思考解决问题的方法和思路，只有这样才能提高实战的操作能力。

第 2 章

◀ Oracle 的安装与配置 ▶

在 Windows 平台下安装 Oracle 12c，图形化的安装包提供了详细的安装向导，通过向导，读者可以一步一步地完成对 Oracle 的安装。本章将主要讲述 Windows 平台下 Oracle 的安装和配置过程，最后讲解 Oracle 的完全卸载方法。

- 掌握如何在 Windows 平台下安装和配置 Oracle 12c
- 掌握启动服务并登录 Oracle 12c 数据库
- 掌握 Oracle 的配置方法
- 熟悉 Oracle 常用图形管理工具
- 掌握 Oracle 的完全卸载方法

2.1 安装 Oracle 12c

安装 Oracle 12c 之前，需要到 Oracle 官方网站（www.oracle.com）去下载该数据库软件，根据不同的系统，下载不同的 Oracle 版本，这里选择 Windows x64 系统的版本，如图 2-1 所示。当然在下载前，需要选中【Accept License Agreement】单选按钮。

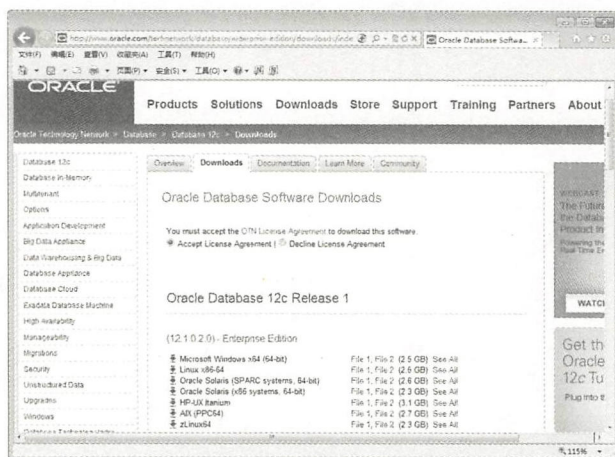


图 2-1 Oracle 下载界面

要想在 Windows 中运行 Oracle 12c 的 64 位版，需要 64 位 Windows 操作系统。本书的操



作系统为 Window 7 的 64 位版本。Windows 可以将 Oracle 服务器作为服务来运行，通常在安装时需要具有系统的管理员权限。

Oracle 下载完成后，找到下载文件，双击进行安装，具体操作步骤如下。

- 步骤 01** 双击下载的 setup.exe 文件，软件会加载并初步校验系统是否可以达到数据库安装的最低配置，如图 2-2 所示。

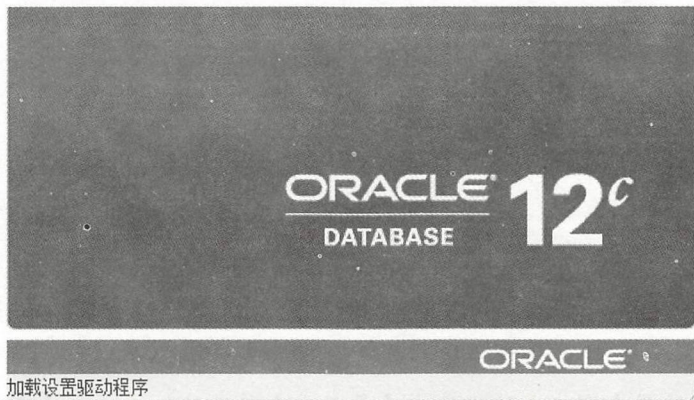


图 2-2 检查操作系统

- 步骤 02** 弹出 Oracle 12c 的【配置安全更新】窗口，如图 2-3 所示，取消选中【我希望通过 My Oracle Support 接收安全更新】复选框，单击【下一步】按钮。

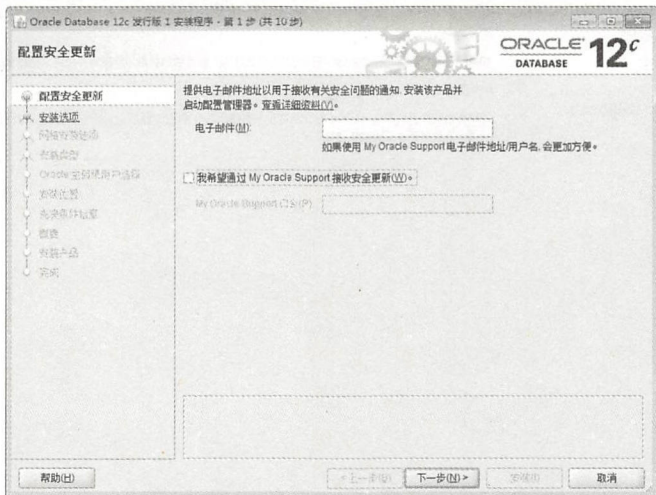


图 2-3 Oracle 12c 的【配置安全更新】窗口

提示

安装时操作系统需要连接网络，如果提示软件更新，选择软件更新即可。

- 步骤 03** 打开【选择安装选项】窗口，选中【创建和配置数据库】单选按钮，单击【下一步】按钮，如图 2-4 所示。

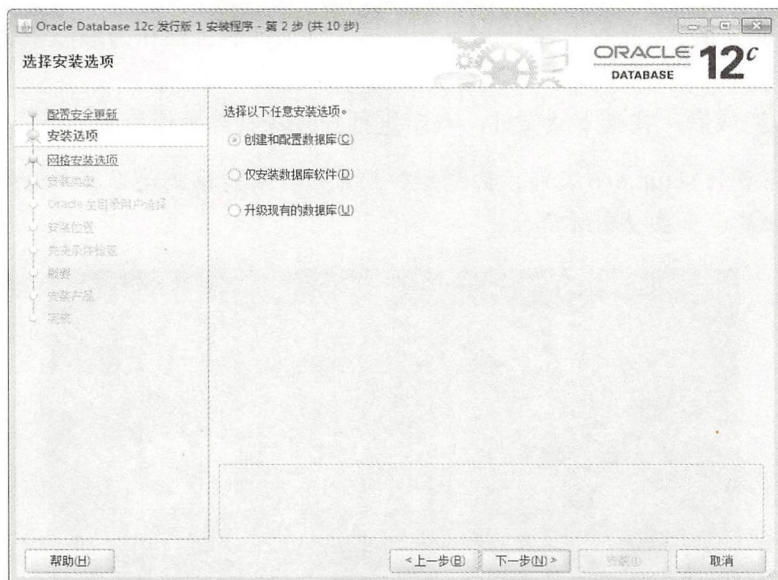


图 2-4 【选择安装选项】窗口

步骤 04 打开【系统类】窗口，这里选中【桌面类】单选按钮，单击【下一步】按钮，如图 2-5 所示。

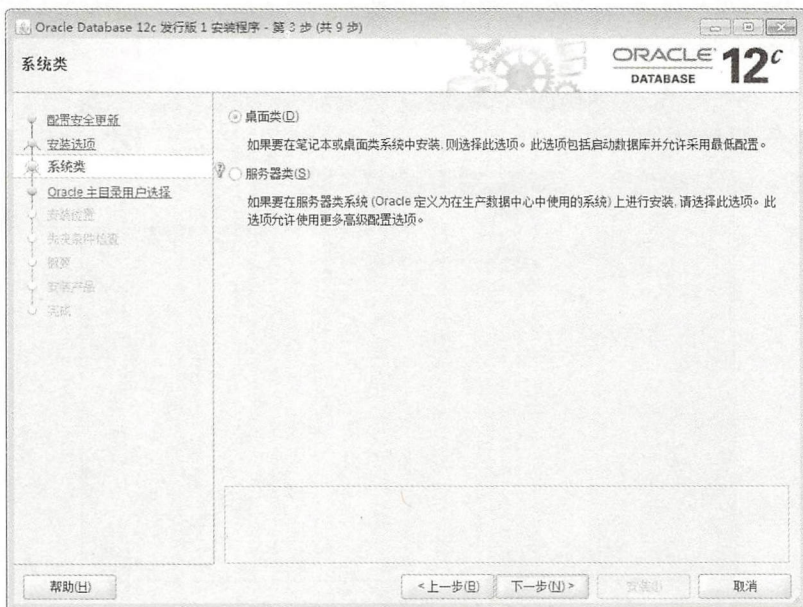


图 2-5 【系统类】窗口

提示

如果选中【服务器类】单选按钮，则用户需要高级的设置。

步骤 05 打开【指定 Oracle 主目录用户】窗口，这一步是其他版本没有的，主要的作用是更安全管理数据库，防止登录 Windows 操作系统的用户误删除了 Oracle 文件。这里选中【创建新 Windows 用户】单选按钮，然后输入用户名和口令，专门管理 Oracle 文



件，然后单击【下一步】按钮，如图 2-6 所示。

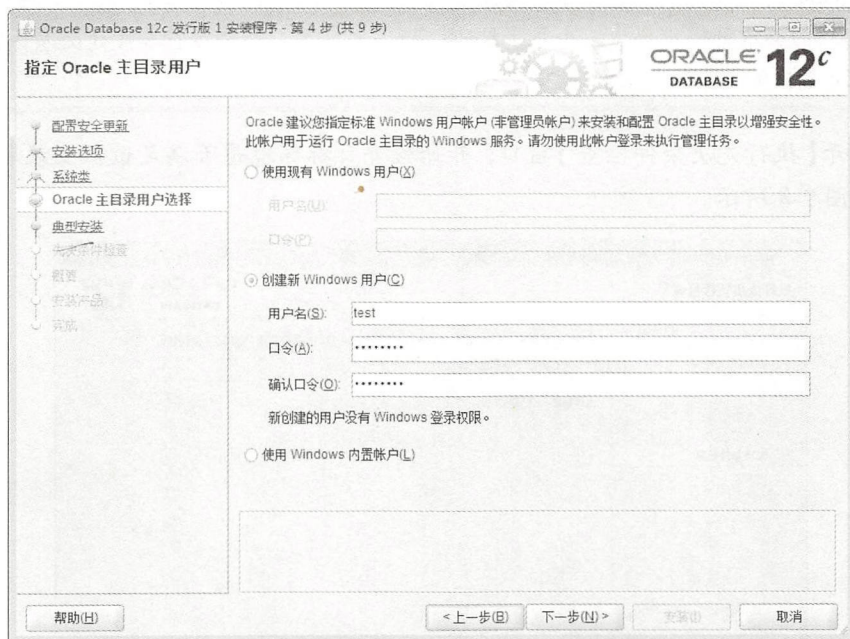


图 2-6 【指定 Oracle 主目录用户】窗口

步骤 06 打开【典型安装配置】窗口，选择 Oracle 的基目录，选择【企业版】和【默认值】，并输入统一的管理口令，单击【下一步】按钮，如图 2-7 所示。

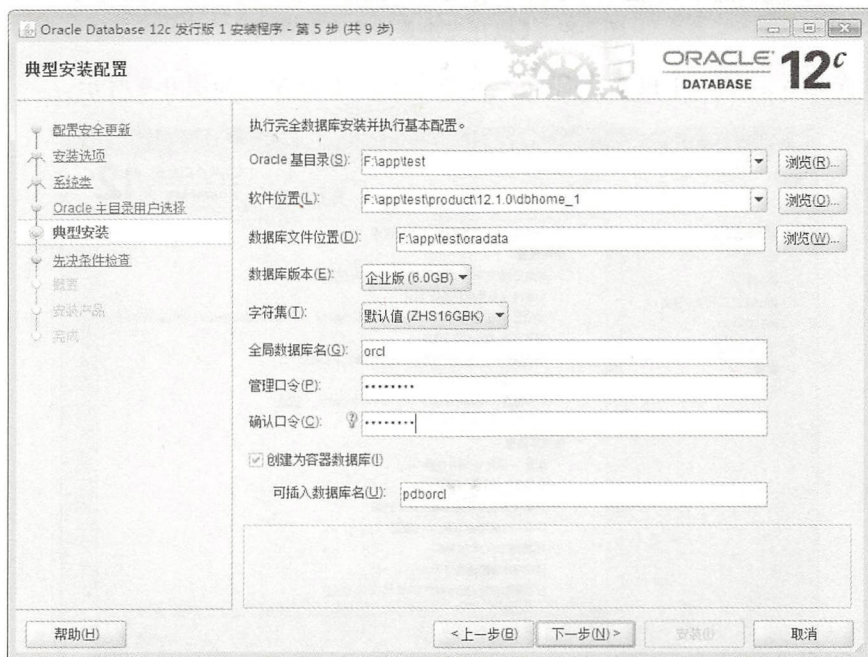


图 2-7 【典型安装配置】窗口



提示

Oracle 为了安全起见，要求密码强度比较高，输入的密码 Oracle 认为不能复制，Oracle 建议的标准密码组合为：小写字母+数字+大写字母，当然字符长度还必须保持在 Oracle 12c 数据库要求的范围之内。

步骤 07 打开【执行先决条件检查】窗口，开始检查目标环境是否满足最低安装和配置要求，如图 2-8 所示。

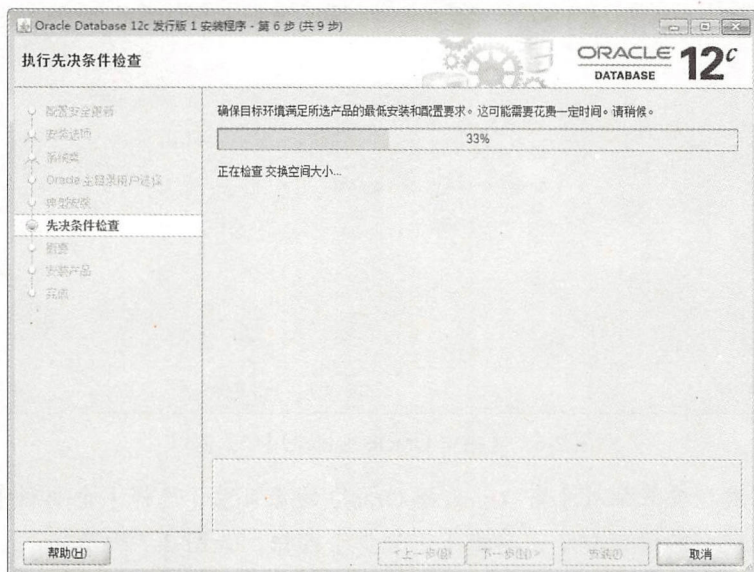


图 2-8 【执行先决条件检查】窗口

步骤 08 检查完成后进入【概要】窗口，单击【安装】按钮，如图 2-9 所示。

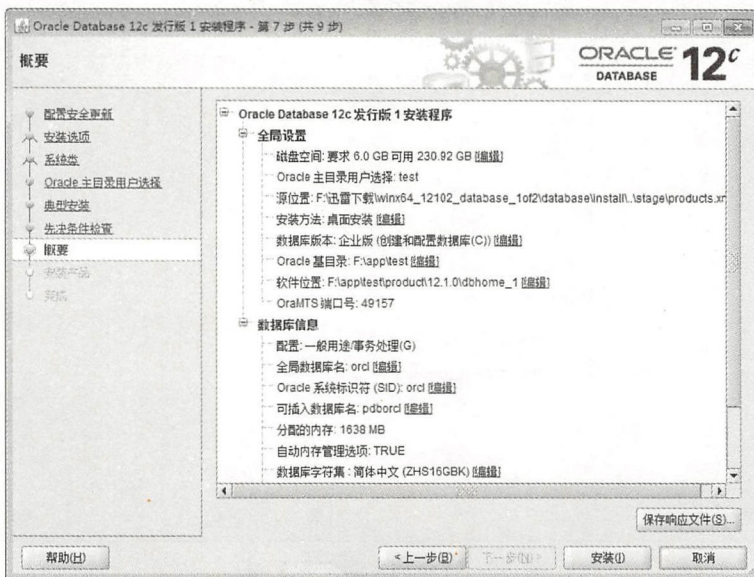


图 2-9 【概要】窗口



步骤 09 进入【安装产品】窗口，开始安装 Oracle 文件，并显示具体内容和进度，如图 2-10 所示。

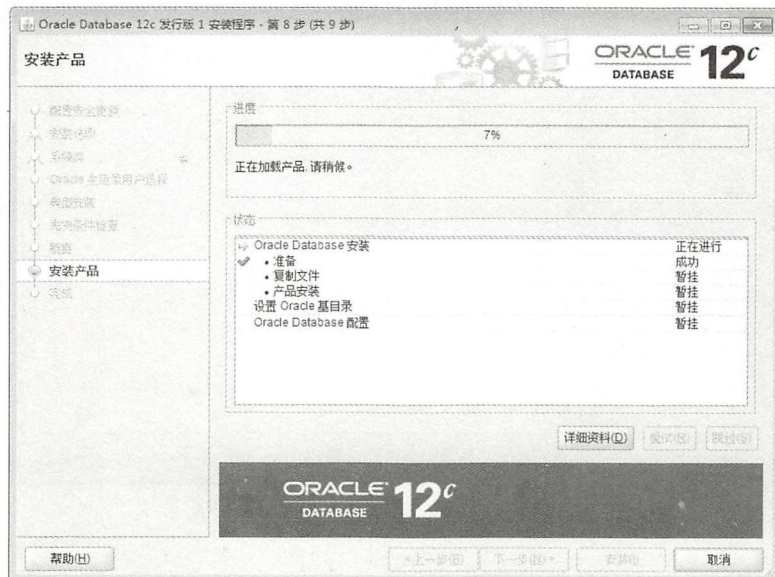


图 2-10 【安装产品】窗口

步骤 10 数据库安装成功后，打开【口令管理】窗口，单击【口令管理】按钮，即可修改管理员的密码，这里修改管理员 SYSTEM 的密码为“Tianyi123456”、超级管理员 SYS 的密码为 TIAN_yi123，设置完成后，单击【确定】按钮即可。如图 2-11 所示。

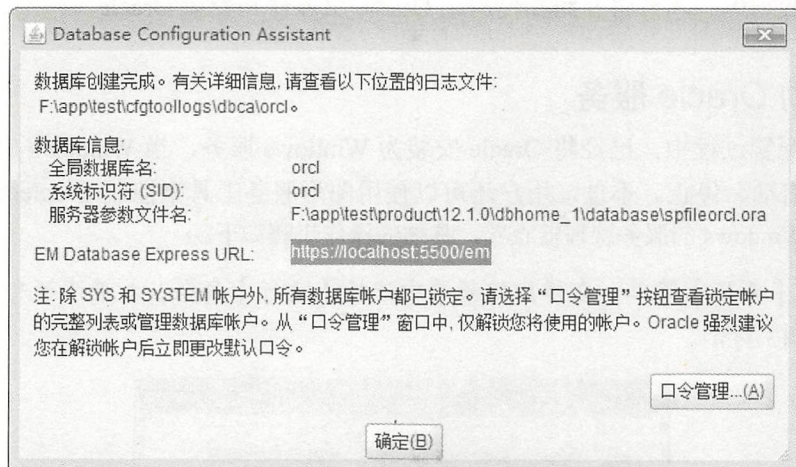


图 2-11 【口令管理】窗口

步骤 11 安装完成后，单击【关闭】按钮，如图 2-12 所示。

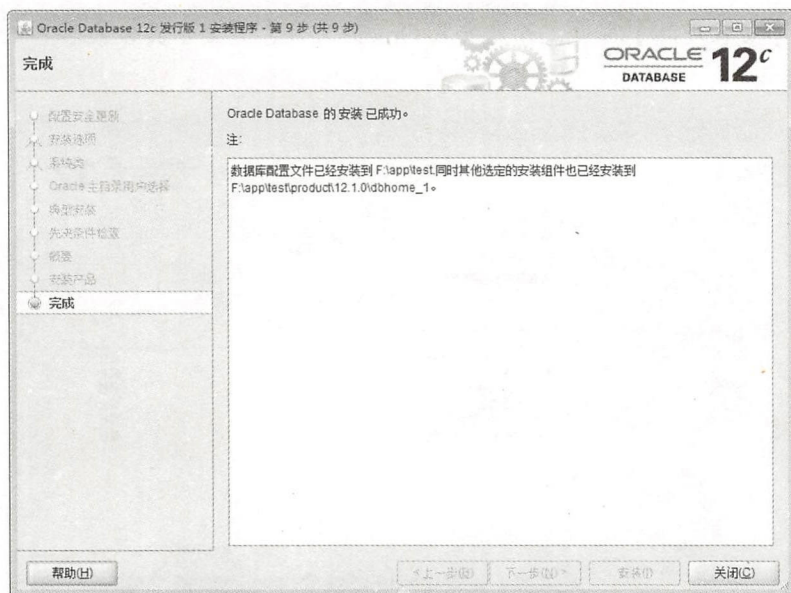


图 2-12 【完成】窗口

2.2 启动服务并登录 Oracle 数据库

Oracle 安装完毕之后，需要启动服务器进程，否则客户端无法连接数据库，更无法通过命令行工具登录数据库。本节将介绍如何启动 Oracle 服务器和登录 Oracle。

2.2.1 启动 Oracle 服务

在前面的配置过程中，已经将 Oracle 安装为 Windows 服务，当 Windows 启动、停止时，Oracle 也自动启动、停止。不过，用户还可以使用图形服务工具来控制 Oracle 服务器。

可以通过 Windows 的服务管理器查看，具体的操作步骤如下。

步骤 01 单击【开始】菜单，在弹出的菜单中选择【运行】命令，打开【运行】对话框，如图 2-13 所示。

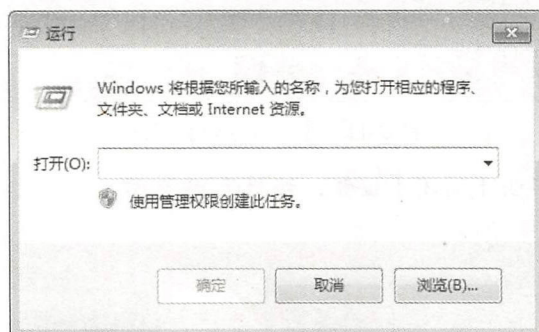


图 2-13 【运行】对话框



步骤 02 在【打开】文本框中输入“services.msc”，单击【确定】按钮，打开 Windows 的服务管理器，在其中可以看到服务名以“Oracle”开头的 5 个服务项，其右边状态全部为“已启动”，表明该服务已经启动，如图 2-14 所示。

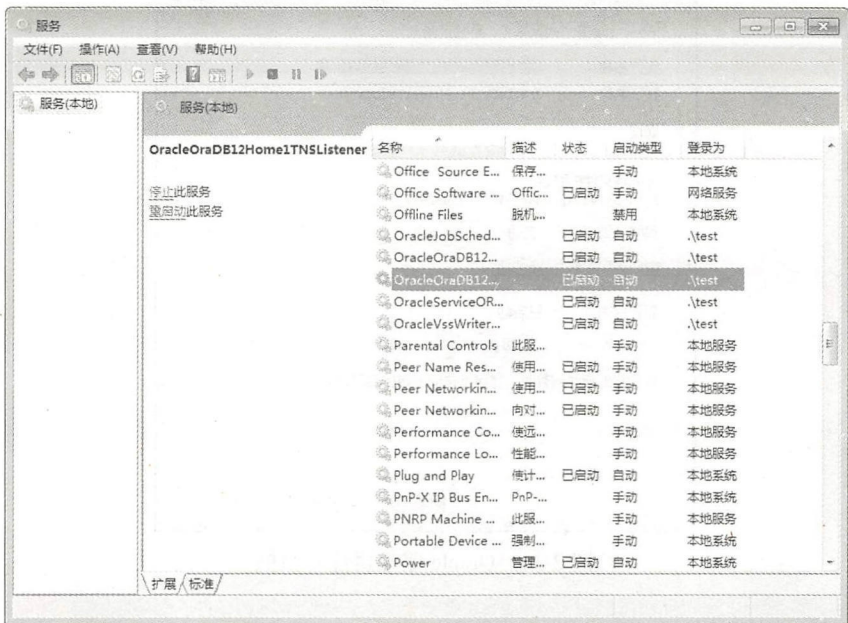


图 2-14 服务管理器窗口

由于设置了 Oracle 为自动启动，在这里可以看到，服务已经启动，而且启动类型为自动。如果没有“已启动”字样，说明 Oracle 服务未启动。此时可以选择服务右击，在弹出的快捷菜单中选择【启动】命令即可，如图 2-15 所示。

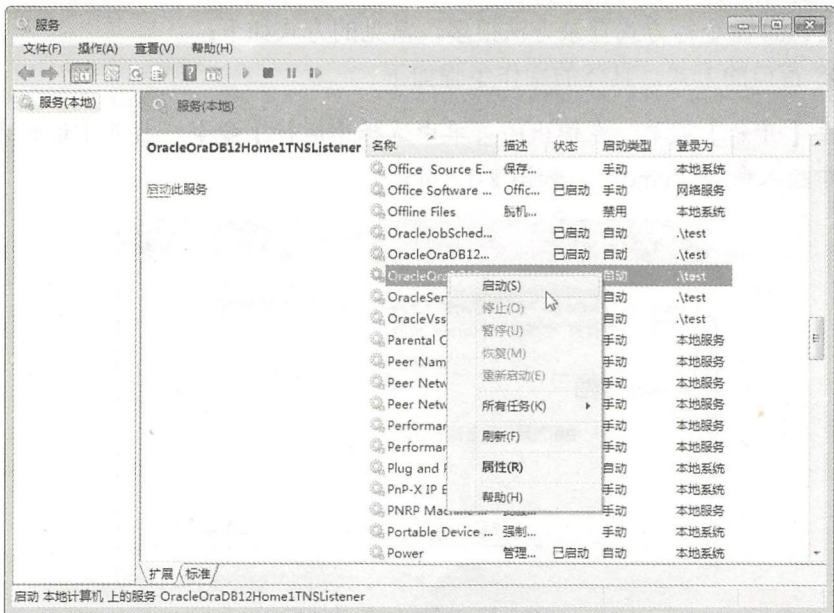


图 2-15 启动 Oracle 服务



也可以直接双击 Oracle 服务，在打开的对话框中通过单击【启动】或【停止】按钮来更改服务状态，如图 2-16 所示。

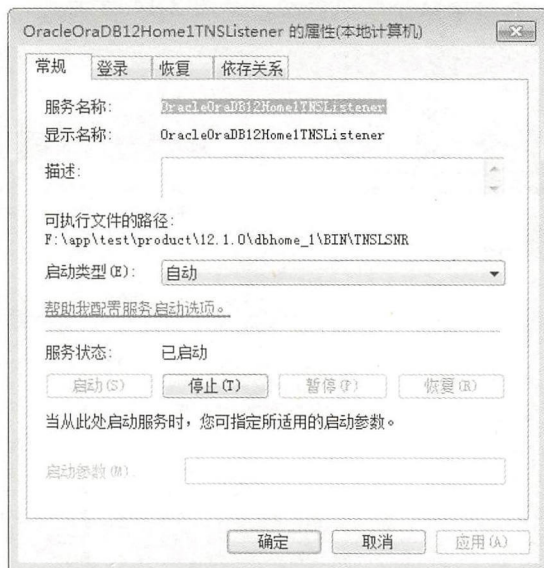


图 2-16 Oracle 服务属性对话框

2.2.2 登录 Oracle 数据库

当 Oracle 服务启动完成后，便可以通过客户端来登录 Oracle 数据库。在 Windows 操作系统下，可以通过两种方式登录 Oracle 数据库。

1. 以 SQL Plus 命令行方式登录

通过 SQL Plus 命令行方式登录方法很多，常见的方式如下：

通过 DOS 窗口的方式，具体的操作步骤如下。

步骤 01 单击【开始】菜单，在弹出的菜单中选择【运行】命令，打开【运行】对话框，在其中输入命令“cmd”，如图 2-17 所示。

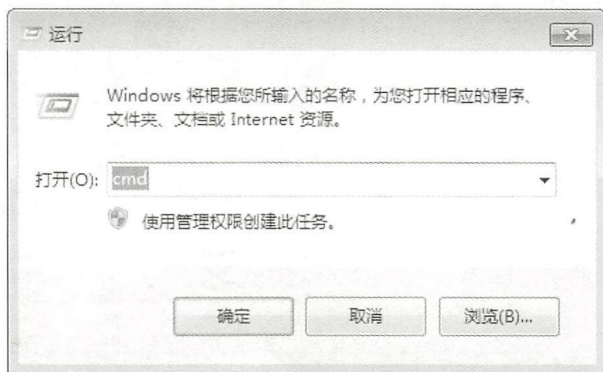


图 2-17 【运行】对话框



步骤 02 单击【确定】按钮，打开 DOS 窗口，输入以下命令并按【Enter】键确认，如图 2-18 所示。

```
sqlplus "/as sysdba"
```

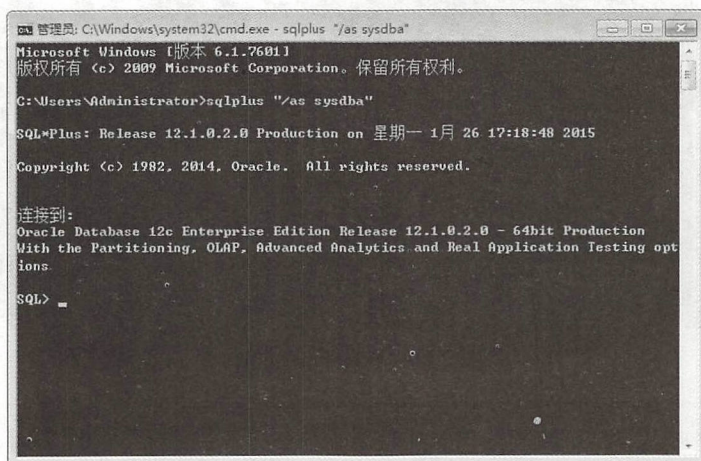


图 2-18 DOS 窗口

直接利用 SQL Plus 登录，具体的操作步骤如下。

步骤 01 依次选择【开始】|【所有程序】|【Oracle-OraDB12Home1】|【应用程序开发】|【SQL Plus】菜单命令，如图 2-19 所示。

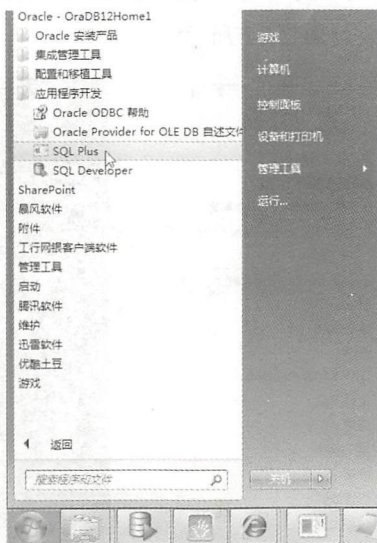


图 2-19 选择【SQL Plus】菜单命令

步骤 02 打开 SQL Plus 窗口，输入用户名和口令并按【Enter】键确认，如图 2-20 所示。

请输入用户名: sys

输入口令: 安装时密码 as sysdba

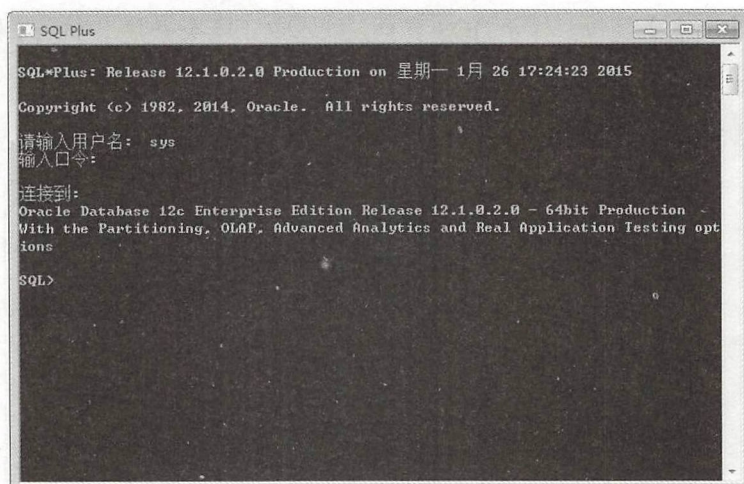


图 2-20 SQL Plus 窗口

提示

当窗口中出现如图 2-20 所示的说明信息，命令提示符变为“SQL>”时，表明已经成功登录 Oracle 服务器了，可以开始对数据库进行操作。

2. 使用 SQL Developer 登录

具体操作步骤如下：

步骤 01 依次选择【开始】|【所有程序】|【Oracle-OraDB12Home1】|【应用程序开发】|【SQL Developer】菜单命令，如图 2-21 所示。

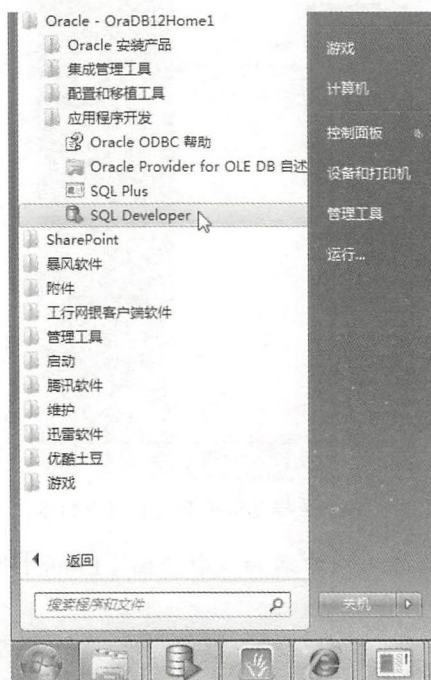


图 2-21 选择【SQL Developer】菜单命令



步骤 02 打开【新建/选择数据库连接】对话框，输入【连接名】，选择【连接类型】为【本地/继承】，选择【角色】为【SYSDBA】，选中【操作系统验证】复选框，单击【连接】按钮，如图 2-22 所示。

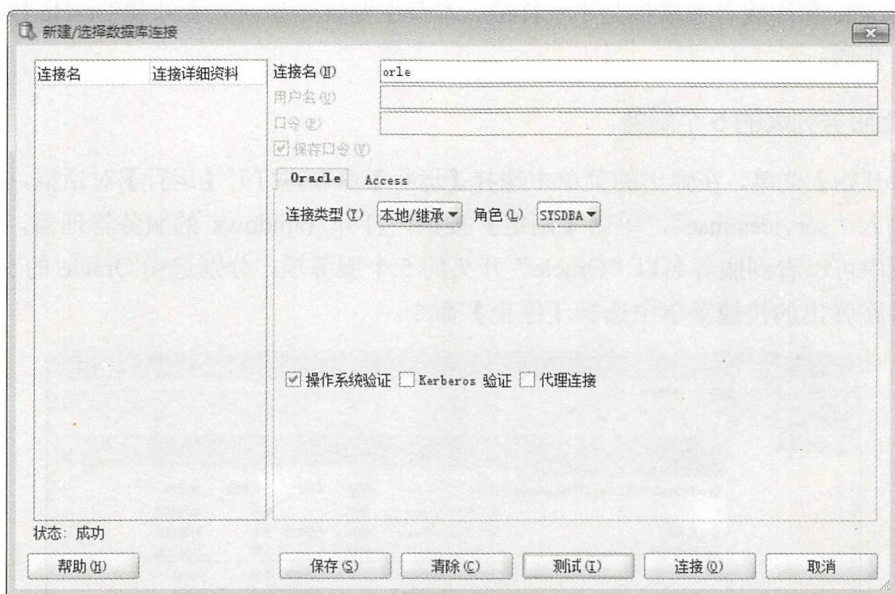


图 2-22 【新建/选择数据库连接】对话框

步骤 03 打开 SQL Developer 主界面窗口，在其中输入 SQL 命令进行相关的操作即可，如图 2-23 所示。

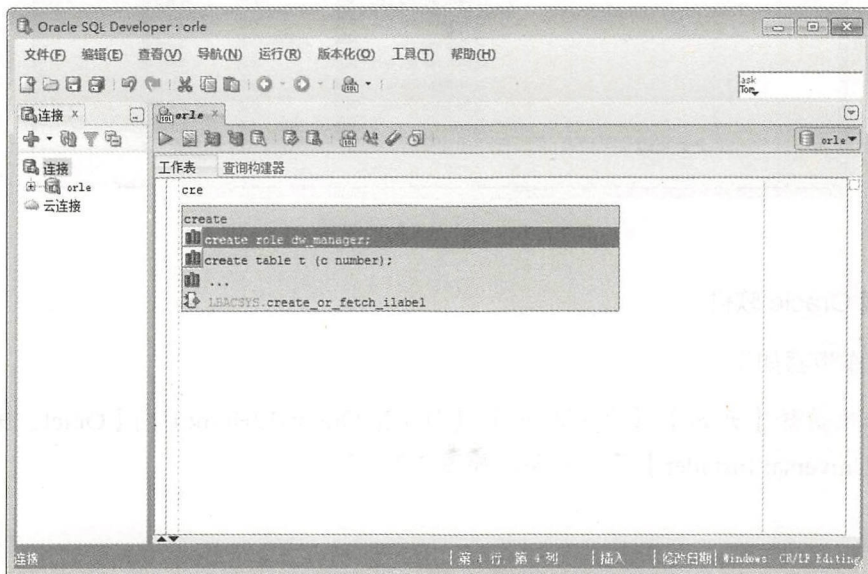


图 2-23 SQL Developer 主界面窗口



2.3 卸载 Oracle 12c

卸载 Oracle 和卸载普通软件是不一样的，本节主要讲述如何完全卸载 Oracle 12c。主要分为以下几个步骤：

1. 停止服务列表的 5 个服务

单击【开始】菜单，在弹出的菜单中选择【运行】命令，打开【运行】对话框，在【打开】文本框中输入“services.msc”，单击【确定】按钮，打开 Windows 的服务管理器，如图 2-24 所示。在其中可以看到服务名以“Oracle”开头的 5 个服务项，分别选中 Oracle 的 5 个服务名称，右击并在弹出的快捷菜单中选择【停止】命令。

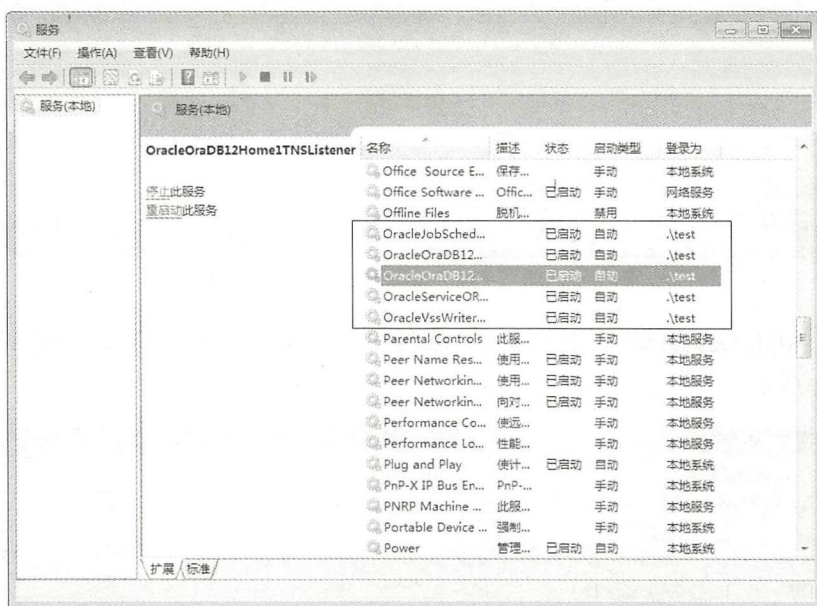


图 2-24 服务管理器窗口

2. 卸载 Oracle 软件

具体操作步骤如下：

步骤 01 依次选择【开始】|【所有程序】|【Oracle-OraDB12Home1】|【Oracle 安装产品】|【Universal Installer】菜单命令，如图 2-25 所示。

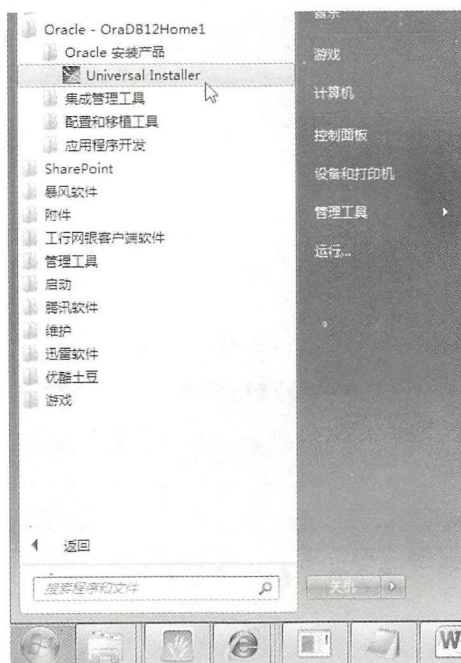


图 2-25 选择【Universal Installer】菜单命令

步骤 02 打开【Oracle Universal Installer: 欢迎使用】对话框，单击【卸载产品】按钮，如图 2-26 所示。

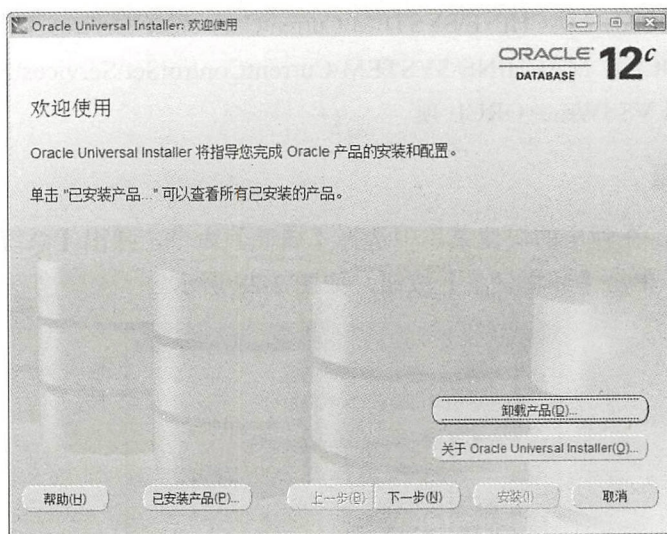


图 2-26 【Oracle Universal Installer: 欢迎使用】对话框

步骤 03 打开【产品清单】对话框，全部选择需要删除的内容，单击【删除】按钮即可开始卸载，如图 2-27 所示。

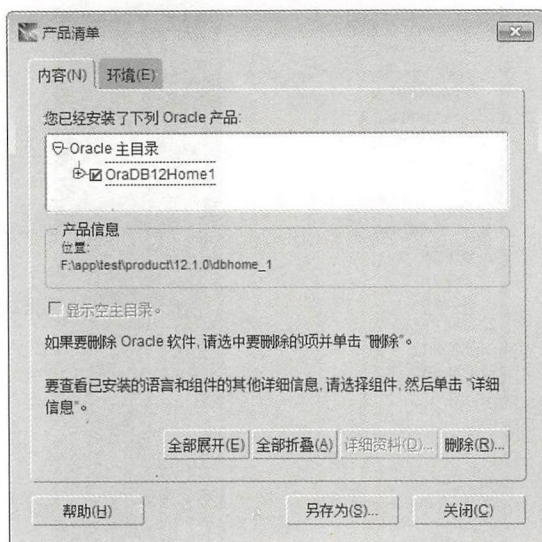


图 2-27 【产品清单】对话框

3. 删除注册表项

在【运行】对话框中输入“regedit”，启动注册表。要彻底删除 Oracle 12c，还需要把注册表中关于 Oracle 的相关信息删除。需要删除的注册表项如下：

- (1) HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE 项
- (2) HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services 节点下的所有 Oracle 项
- (3) HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application 节点下的所有 Oracle.VSSWriter.ORCL 项

4. 删除环境变量

右击【计算机】，在弹出的快捷菜单中选择【属性】命令，弹出【系统属性】对话框，选择【高级】选项卡，单击【环境变量】按钮，如图 2-28 所示。

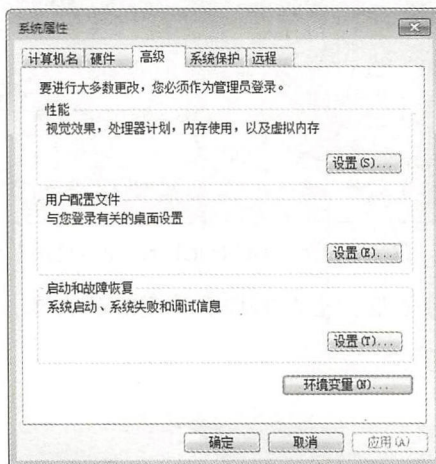


图 2-28 【系统属性】对话框



打开【环境变量】对话框，在【系统变量】中查找【Path】变量，然后删除即可，如果发现另外关于 Oracle 的选项，一并删除即可，如图 2-29 所示。

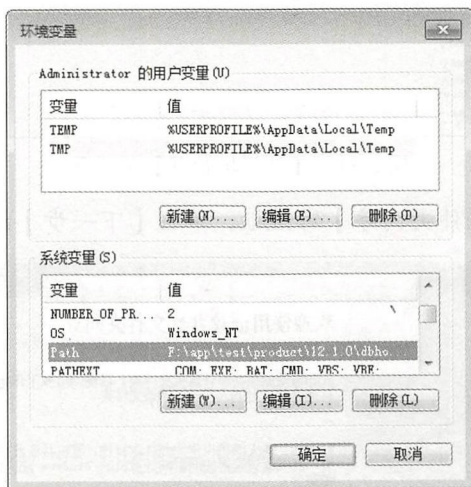


图 2-29 【环境变量】对话框

2.4 疑难解惑

计算机技术具有很强的操作性，Oracle 的安装和配置是一件非常简单的事，但是在操作过程中也可能出现问题，读者需要多实践、多总结。

疑问 1：无法安装 Oracle 12c 软件安装包，提示对话框如图 2-30 所示，如何解决？

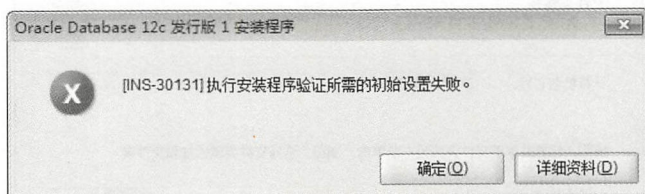


图 2-30 无法安装提示对话框

解决上述问题的具体操作步骤如下：

- 步骤 01** 右击【计算机】，在弹出的快捷菜单中选择【管理】命令，弹出【计算机管理】对话框，在左侧列表中选择【系统工具】|【共享文件夹】，选择【共享】右击，在弹出的快捷菜单中选择【新建共享】命令，如图 2-31 所示



图 2-31 【计算机管理】对话框

步骤 02 打开【创建共享文件夹向导】对话框，单击【下一步】按钮，如图 2-32 所示。

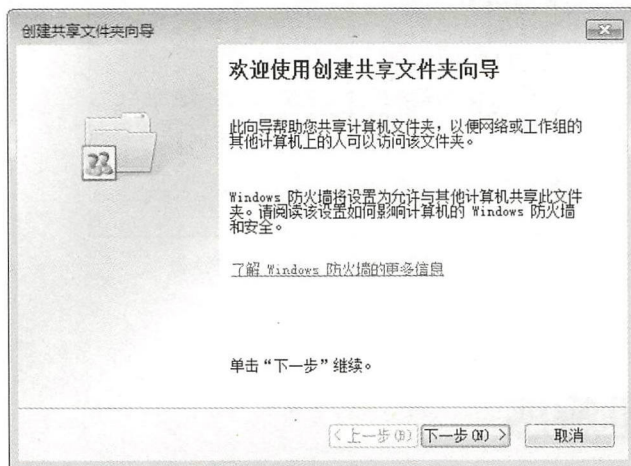


图 2-32 【创建共享文件夹向导】对话框

步骤 03 打开【文件夹路径】对话框，选择 C 盘为文件夹的路径，如图 2-33 所示。

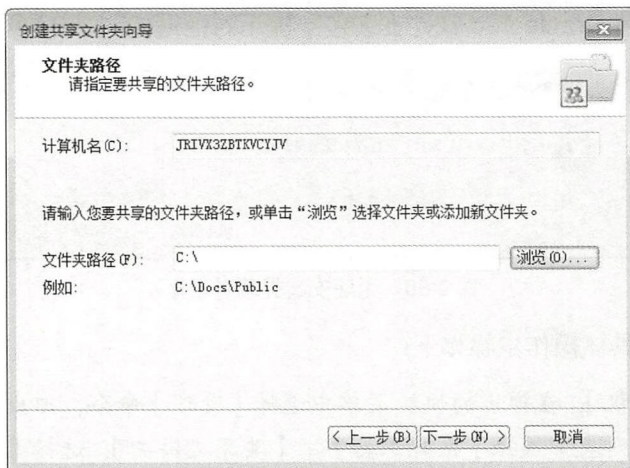


图 2-33 【文件夹路径】对话框

步骤 04 打开【名称、描述和设置】对话框，在【共享名】文本框中输入“C\$”，单击【下一步】按钮，如图 2-34 所示。

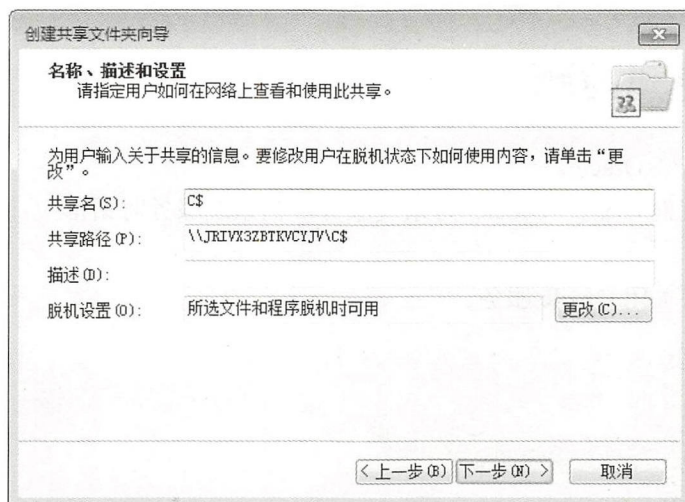


图 2-34 【名称、描述和设置】对话框

步骤 05 打开【共享文件夹的权限】对话框，选中【管理员有完全访问权限；其他用户有只读权限】单选按钮，单击【完成】按钮，如图 2-35 所示

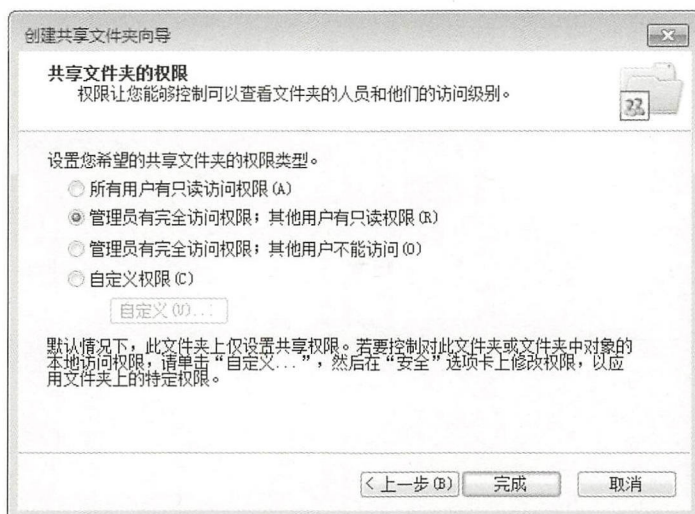


图 2-35 【共享文件夹的权限】对话框

疑问 2： Oracle 12c 卸载完成后，仍然无法安装 Oracle 12c，怎么办？

按照 Oracle 12c 安装书中的 4 个步骤去卸载，为了更加彻底删除 Oracle，还需要把安装目录下的内容全部删除，删除后还需要重新启动计算机，这样就可以把 Oracle 完全删除了，最后才能重新安装 Oracle。

2.5 经典习题

- (1) 下载并安装 Oracle。
- (2) 使用配置向导配置 Oracle 为系统服务，在系统服务对话框中，手动启动或者关闭 Oracle 服务。
- (3) 启动或者关闭 Oracle 服务。

第 3 章

数据库和数据表的基本操作

Oracle 安装好以后，用户可以创建和删除数据库。在数据库中，数据表是最重要、最基本的操作对象，是数据存储的基本单位。数据表被定义为列的集合，数据在表中是按照行和列的格式来存储的。每一行代表一条唯一的记录，每一列代表记录中的一个域。

本章将详细介绍数据表的基本操作，主要内容包括：创建数据表、查看数据表结构、修改数据表、删除数据表。通过本章的学习，读者能够熟练掌握数据表的基本概念，理解约束、默认和规则的含义并且学会运用；能够在图形界面模式和命令行模式下熟练地完成有关数据表的常用操作。

- 掌握如何创建数据库
- 熟悉数据库的删除操作
- 掌握如何创建数据表
- 掌握查看数据表结构的方法
- 掌握如何修改数据表
- 熟悉删除数据表的方法
- 熟练操作综合案例数据表的基本操作

3.1 创建数据库

Oracle 12c 在安装过程中已经创建了名称为 orle 的数据库。用户也可以在安装完成后重新创建数据库，具体操作步骤如下：

步骤 01 依次选择【开始】|【所有程序】|【Oracle-OraDB12Home1】|【配置和移植工具】|【Database Configuration Assistant】菜单命令，如图 3-1 所示。

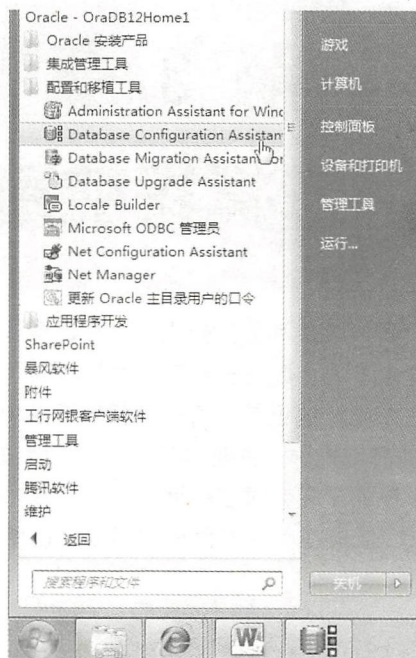


图 3-1 选择【Database Configuration Assistant】菜单命令

步骤 02 打开【数据库操作】窗口，选中【创建数据库】单选按钮，如图 3-2 所示。

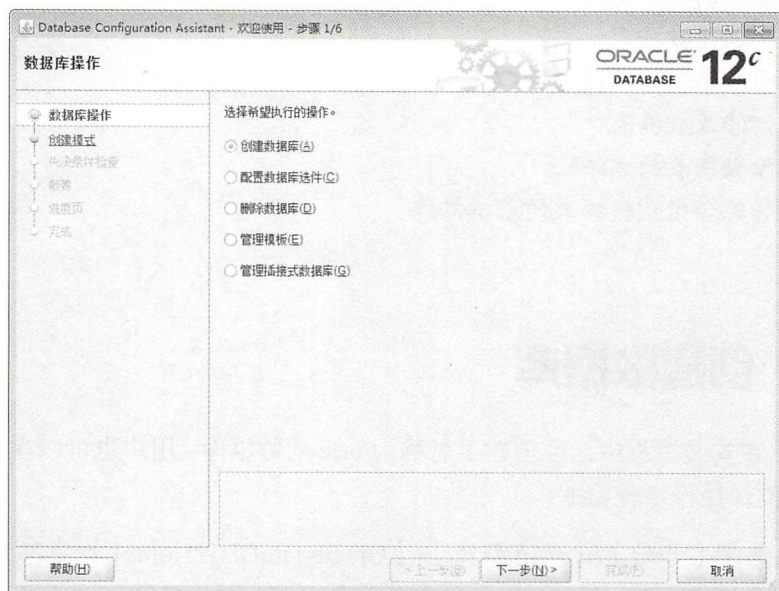


图 3-2 【数据库操作】窗口

步骤 03 打开【创建模式】窗口，输入全局数据库的名称、设置数据库文件的位置、输入管理口令和 test 用户口令，然后单击【下一步】按钮，如图 3-3 所示。

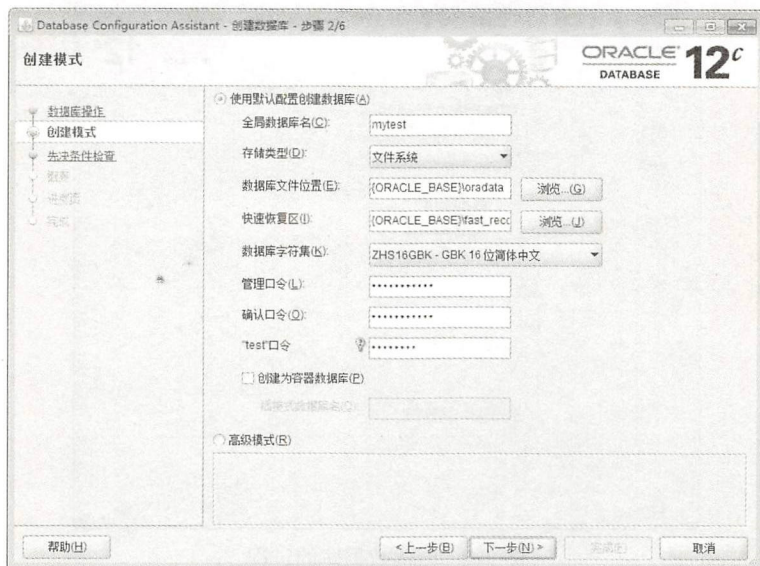


图 3-3 【创建模式】窗口

步骤 04 打开【概要】窗口，查看创建数据库的详细信息，检查无误后，单击【完成】按钮，如图 3-4 所示。

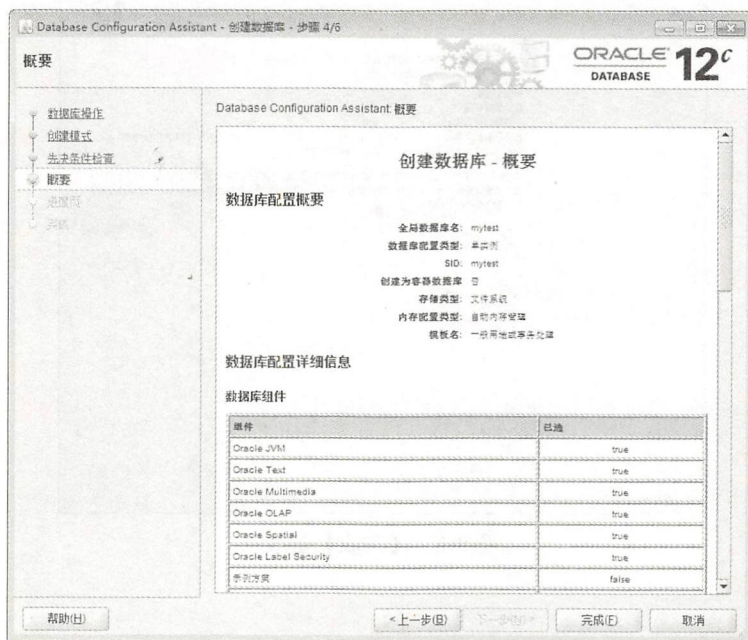


图 3-4 【概要】窗口

步骤 05 系统开始自动创建数据库，并显示数据库的创建过程和创建的详细信息，如图 3-5 所示。

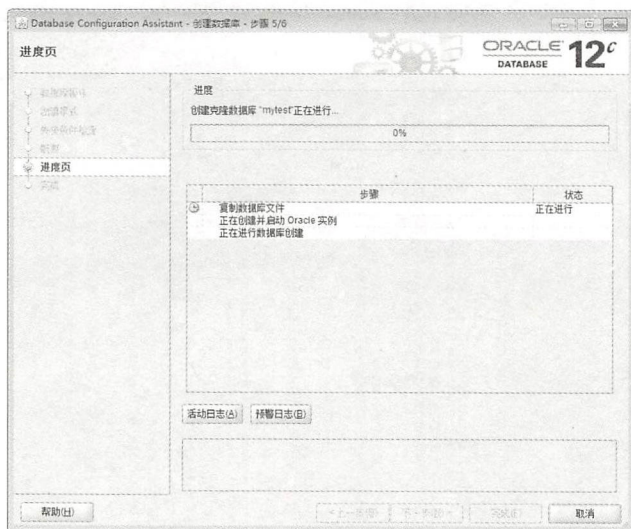


图 3-5 创建数据库的过程

步骤 06 数据库创建完成后，打开【完成】窗口，查看创建数据库的最终信息，单击【关闭】按钮即可完成数据库的创建操作，如图 3-6 所示。

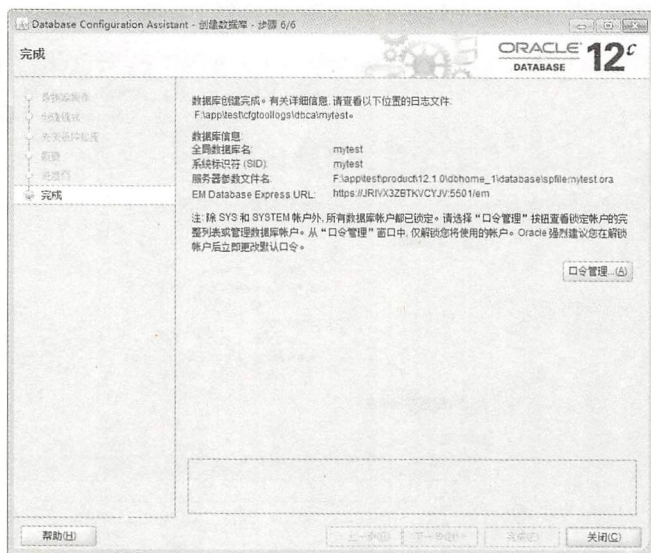


图 3-6 【完成】窗口

3.2 删除数据库

删除数据库是将已经存在的数据库从磁盘空间上清除，清除之后，数据库中的所有数据也将一同被删除。删除数据库的具体操作步骤如下：

步骤 01 依次选择【开始】|【所有程序】|【Oracle-OraDB12Home1】|【配置和移植工具】|

【Database Configuration Assistant】菜单命令，打开【数据库操作】窗口，选中【删除数据库】单选按钮，如图 3-7 所示。

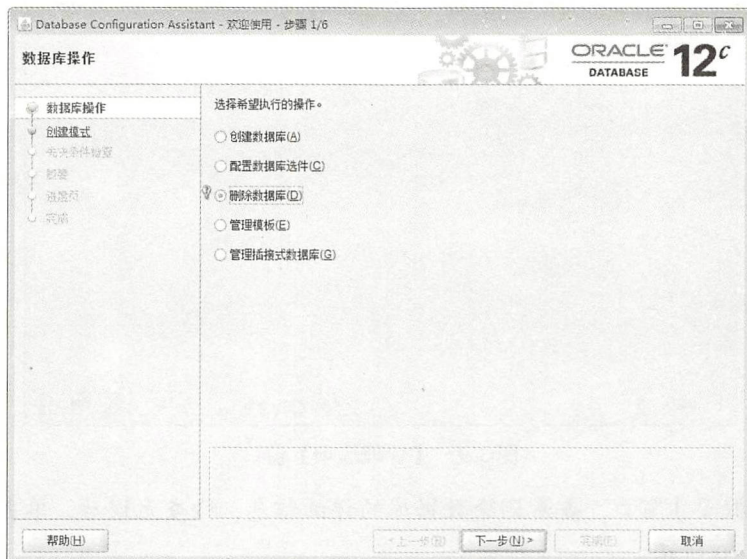


图 3-7 【数据库操作】窗口

步骤 02 打开【删除数据库】窗口，选择需要删除的数据库，本实例选择 MYTEST 数据库，输入数据库管理员的名称和管理口令，单击【下一步】按钮，如图 3-8 所示。

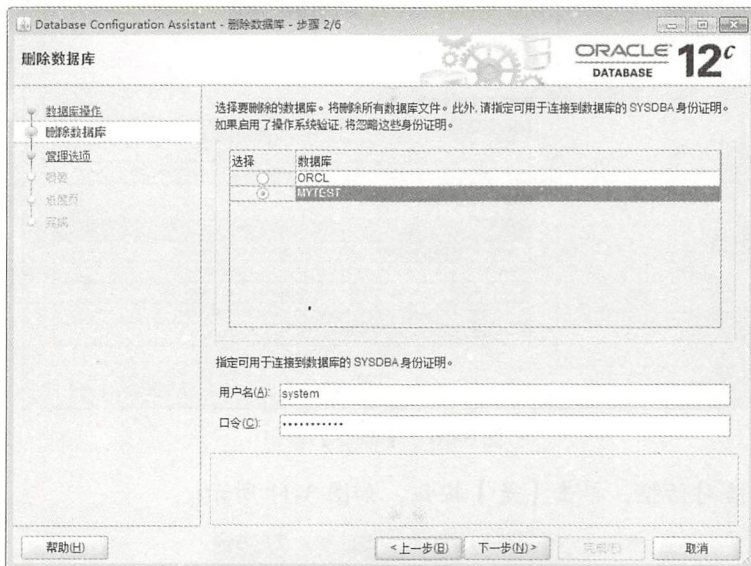


图 3-8 【删除数据库】窗口

步骤 03 打开【管理选项】窗口，单击【下一步】按钮，如图 3-9 所示。

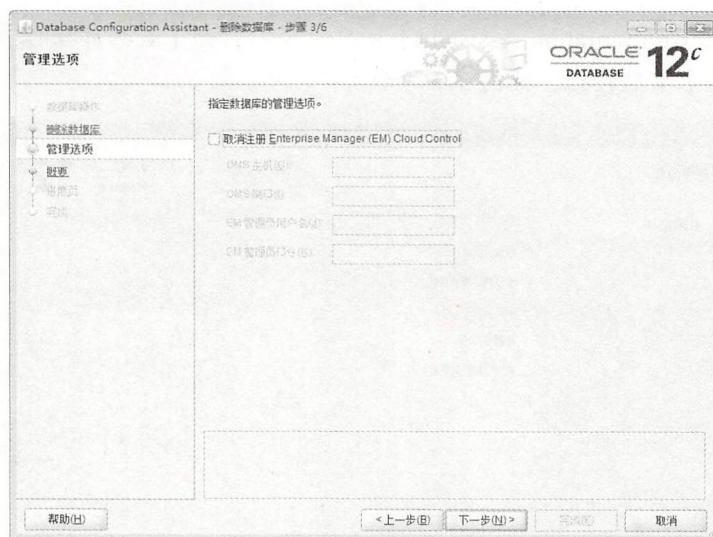


图 3-9 【管理选项】窗口

步骤 04 打开【概要】窗口，查看删除数据库的详细信息，检查无误后，单击【完成】按钮，如图 3-10 所示。

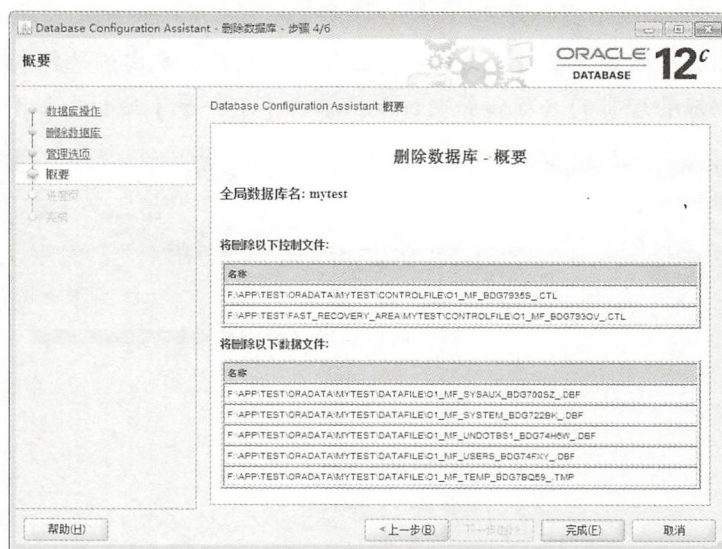


图 3-10 【概要】窗口

步骤 05 弹出警告对话框，单击【是】按钮，如图 3-11 所示。

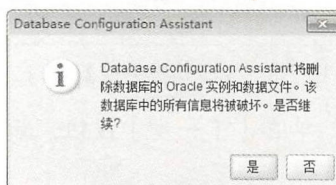


图 3-11 警告对话框

步骤 06 系统开始自动删除数据库，并显示数据库的删除过程和删除的详细信息，如图 3-12 所示。

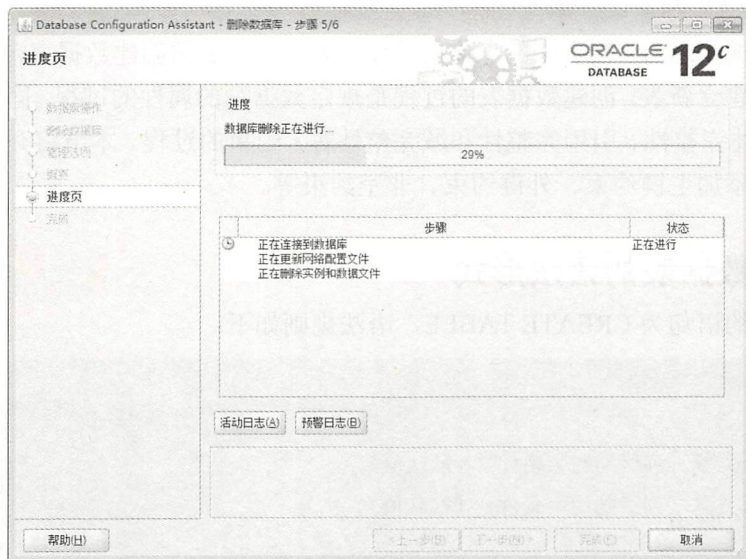


图 3-12 删除数据库的过程

步骤 07 数据库删除完成后，打开【完成】窗口，单击【关闭】按钮即可完成数据库的删除操作，如图 3-13 所示。

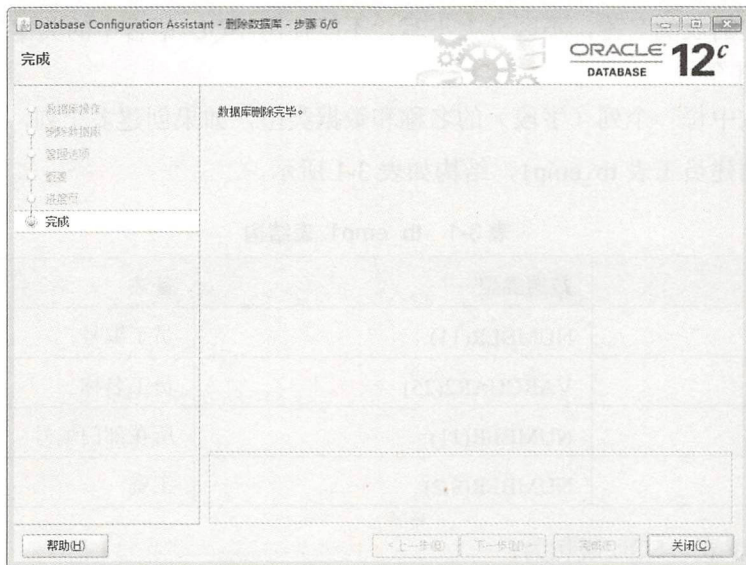


图 3-13 【完成】窗口

提示

执行删除数据库操作时要非常谨慎，因为在执行该操作后，数据库中存储的所有数据表和数据也将一同被删除，而且不能恢复。

3.3 创建数据表

在创建完数据库之后，接下来的工作就是创建数据表。所谓创建数据表，指的是在已经创建好的数据库中建立新表。创建数据表的过程是规定数据列的属性的过程，同时也是实施数据完整性（包括实体完整性、引用完整性和域完整性等）约束的过程。本节将介绍创建数据表的语法形式，如何添加主键约束、外键约束、非空约束等。

3.3.1 创建数据表的语法形式

创建数据表的语句为 CREATE TABLE，语法规则如下：

```
CREATE TABLE <表名>
(
    字段名1, 数据类型 [列级别约束条件] [默认值],
    字段名2, 数据类型 [列级别约束条件] [默认值],
    .....
    [表级别约束条件]
);
```

使用 CREATE TABLE 创建表时，必须指定以下信息：

（1）要创建的表的名称，不区分大小写，不能使用 SQL 语言中的关键字，如 DROP、ALTER、INSERT 等。

（2）数据表中每一个列（字段）的名称和数据类型，如果创建多个列，要用逗号隔开。

【例 3.1】创建员工表 tb_emp1，结构如表 3-1 所示。

表 3-1 tb_emp1 表结构

字段名称	数据类型	备注
id	NUMBER(11)	员工编号
name	VARCHAR2(25)	员工名称
deptId	NUMBER(11)	所在部门编号
salary	NUMBER(9,2)	工资

创建 tb_emp1 表，SQL 语句为：

```
CREATE TABLE tb_emp1
(
    id      NUMBER(11),
    name    VARCHAR2(25),
    deptId  NUMBER(11),
```

```
salary NUMBER(9,2)
);
```

语句执行后，便创建了一个名称为 tb_emp1 的数据表，使用 DESC 表名；语句查看数据表是否创建成功，SQL 语句如下：

```
SQL> DESC tb_emp1;
名称  是否  类型
-----
ID      NUMBER(11)
NAME    VARCHAR2(25)
DEPTID  NUMBER(11)
SALARY  NUMBER(9,2)
```

可以看到，数据库中已经有了数据表 tb_emp1，数据表创建成功。

3.3.2 使用主键约束

主键，又称主码，是表中一行或多列的组合。主键约束（Primary Key Constraint）要求主键列的数据唯一，并且不允许为空。主键能够唯一地标识表中的一条记录，可以结合外键来定义不同数据表之间的关系，并且可以加快数据库查询的速度。主键和记录之间的关系如同身份证和人之间的关系，它们之间是一一对应的。主键分为两种类型：单字段主键和多字段联合主键。

1. 单字段主键

主键由一个字段组成，SQL 语句格式分为以下两种情况。

(1) 在定义列的同时指定主键，语法规则如下：

```
字段名 数据类型 PRIMARY KEY [默认值]
```

【例 3.2】定义数据表 tb_emp2，其主键为 id，SQL 语句如下：

```
CREATE TABLE tb_emp2
(
id      NUMBER(11) PRIMARY KEY,
name    VARCHAR2(25),
deptId  NUMBER(11),
salary  NUMBER(9,2)
);
```

(2) 在定义完所有列之后指定主键，语法规则如下：

```
[CONSTRAINT <约束名>] PRIMARY KEY [字段名]
```


【例 3.3】定义数据表 tb_emp3，其主键为 id，SQL 语句如下：

```
CREATE TABLE tb_emp3
(
  id NUMBER(11),
  name VARCHAR2(25),
  deptId NUMBER(11),
  salary NUMBER(9,2),
  PRIMARY KEY(id)
);
```

上述两个例子执行后的结果是一样的，都会在 id 字段上设置主键约束。

2. 多字段联合主键

主键由多个字段联合组成，语法规则如下：

```
PRIMARY KEY [字段1, 字段2, ..., 字段 n]
```

【例 3.4】定义数据表 tb_emp4，假设表中没有主键 id，为了唯一确定一个员工，可以把 name、deptId 联合起来作为主键，SQL 语句如下：

```
CREATE TABLE tb_emp4
(
  name VARCHAR2(25),
  deptId NUMBER(11),
  salary NUMBER(9,2),
  PRIMARY KEY(name,deptId)
);
```

语句执行后，便创建了一个名称为 tb_emp4 的数据表，name 字段和 deptId 字段组合在一起成为 tb_emp4 的多字段联合主键。

3. 使用 ALTER TABLE 语句为表添加主键约束

在创建表时如果没有添加主键约束，可以在修改表时为表添加主键约束。添加主键约束的语法格式如下：

```
ALTER TABLE 数据表名称
ADD CONSTRAINTS 约束名称 PRIMARY KEY (字段名称)
```

【例 3.5】定义数据表 tb1_emp1，修改其主键为 id，创建数据表的 SQL 语句如下：

```
CREATE TABLE tb1_emp1
(
  id NUMBER(11),
  name VARCHAR2(25),
```

```
deptId NUMBER(11),
salary NUMBER(9,2),
);
```

通过 ALTER TABLE 修改 id 为主键，SQL 语句如下：

```
ALTER TABLE tbl_emp1
ADD CONSTRAINTS pk_id PRIMARY KEY (id);
```

4. 移除主键约束

对于不必要的主键约束，可以将其移除，具体的语法格式如下：

```
ALTER TABLE 数据表名称
DROP CONSTRAINTS 约束名称
```

【例 3.6】移除数据表 tbl_emp1 的主键约束 pk_id，SQL 语句如下：

```
ALTER TABLE tbl_emp1
DROP CONSTRAINTS pk_id;
```

上述语句执行完成后，即可成功移除主键约束 pk_id。

3.3.3 使用外键约束

外键用来在两个表的数据之间建立链接，它可以是一列或者多列。一个表可以有一个或多个外键。外键对应的是参照完整性，一个表的外键可以为空值，若不为空值，则每一个外键值必须等于另一个表中主键的某个值。

外键：首先它是表中的一个字段，它可以不是本表的主键，但对应另外一个表的主键。外键的主要作用是保证数据引用的完整性，定义外键后，不允许删除在另一个表中具有关联关系的行。外键的作用是保持数据的一致性、完整性。例如，部门表 tb_dept1 的主键是 id，在员工表 tb_emp5 中有一个键 deptId 与这个 id 关联。

主表（父表）：对于两个具有关联关系的表而言，相关联字段中主键所在的那个表即是主表。

从表（子表）：对于两个具有关联关系的表而言，相关联字段中外键所在的那个表即是从表。

1. 创建外键

创建外键的语法规则如下：

```
[CONSTRAINT <外键名>] FOREIGN KEY 字段名1 [ , 字段名2, ...]
REFERENCES <主表名> 主键列1 [ , 主键列2, ...]
```

“外键名”为定义的外键约束的名称，一个表中不能有相同名称的外键；“字段名”表示子表需要添加外键约束的字段列；“主表名”即被子表外键所依赖的表的名称；“主键列”表示主表中定义的主键列，或者列组合。

【例 3.7】定义数据表 tb_emp5，并在 tb_emp5 表上创建外键约束。

创建一个部门表 tb_dept1，表结构如表 3-2 所示，SQL 语句如下：

```
CREATE TABLE tb_dept1
(
  id      NUMBER(11) PRIMARY KEY,
  name    VARCHAR2(22) NOT NULL,
  location VARCHAR2(50)
);
```

表 3-2 tb_dept1 表结构

字段名称	数据类型	备注
id	NUMBER(11)	部门编号
name	VARCHAR2(22)	部门名称
location	VARCHAR2(50)	部门位置

定义数据表 tb_emp5，让它的键 deptId 作为外键关联到 tb_dept1 的主键 id，SQL 语句为：

```
CREATE TABLE tb_emp5
(
  id      NUMBER(11) PRIMARY KEY,
  name    VARCHAR2(25),
  deptId  NUMBER(11),
  salary  NUMBER(9,2),
  CONSTRAINT fk_emp_dept1 FOREIGN KEY(deptId) REFERENCES tb_dept1(id)
);
```

以上语句执行成功之后，在表 tb_emp5 上添加了名称为 fk_emp_dept1 的外键约束，外键名称为 deptId，其依赖于表 tb_dept1 的主键 id。

提示

关联指的是在关系型数据库中，相关表之间的联系。它是通过相容或相同的属性或属性组来表示的。子表的外键必须关联父表的主键，且关联字段的数据类型必须匹配，如果类型不一样，则创建子表时，就会出现错误。

2. 在修改数据表时添加外键约束

在创建表时如果没有添加外键约束，可以在修改表时为表添加外键约束。添加外键约束的语法格式如下：

```
ALTER TABLE 数据表名称
ADD CONSTRAINTS 约束名称 FOREIGN KEY (外键约束的字段名称)
REFERENCE 数据表名称 (字段名称)
ON DELETE CASCADE;
```

【例 3.8】在 tb_emp5 表上添加外键约束，SQL 语句如下：

```
ALTER TABLE tb_emp5
ADD CONSTRAINTS fk_emp_dept1 FOREIGN KEY (deptId)
REFERENCE tb_dept1(id)
ON DELETE CASCADE;
```

语句执行完成后，即为 tb_emp5 表的 deptId 字段添加了外键约束。

3. 移除外键约束

对于不需要的外键约束，可以将其移除，具体的语法格式如下：

```
ALTER TABLE 数据表名称
DROP CONSTRAINTS 约束名称
```

【例 3.9】移除数据表 tb_emp5 的外键约束 fk_emp_dept1，SQL 语句如下：

```
ALTER TABLE tb_emp5
DROP CONSTRAINTS fk_emp_dept1;
```

上述语句执行完成后，即可成功移除 tb_emp5 的外键约束。

3.3.4 使用非空约束

非空约束 (Not Null Constraint) 指字段的值不能为空。对于使用了非空约束的字段，如果用户在添加数据时没有指定值，数据库系统会报错。

1. 创建非空约束

非空约束的语法规则如下：

```
字段名 数据类型 not null
```

【例 3.10】定义数据表 tb_emp6，指定员工的名称不能为空，SQL 语句如下：

```
CREATE TABLE tb_emp6
(
id    NUMBER(11) PRIMARY KEY,
name  VARCHAR2(25) NOT NULL,
deptId NUMBER(11),
salary NUMBER(9,2)
);
```

执行后，在 tb_emp6 中创建了一个 name 字段，其插入值不能为空 (NOT NULL)。

2. 修改表时添加非空约束

在创建表时如果没有添加非空约束，可以在修改表时为表添加非空约束。添加非空约束



的语法格式如下：

```
ALTER TABLE 数据表名称  
MODIFY 字段名称 NOT NULL;
```

【例 3.11】将 tb_emp5 表上的字段 name 指定为不能为空，SQL 语句如下：

```
ALTER TABLE tb_emp5  
MODIFY name NOT NULL;
```

语句执行完成后，即为 tb_emp5 表的 name 字段添加了非空约束。

3. 移除非空约束

对于不需要的非空约束，可以将其移除，具体的语法格式如下：

```
ALTER TABLE 数据表名称  
MODIFY 字段名称 NULL;
```

【例 3.12】移除数据表 tb_emp5 的非空约束，SQL 语句如下：

```
ALTER TABLE tb_emp5  
MODIFY name NULL;
```

上述语句执行完成后，即可成功移除非空约束。

3.3.5 使用唯一性约束

唯一性约束（Unique Constraint）要求该列唯一，允许为空，但只能出现一个空值。唯一约束可以确保一列或者几列不出现重复值。

1. 创建唯一性约束

唯一性约束的语法规则如下：

（1）在定义完列之后直接指定唯一性约束，语法规则如下：

```
字段名 数据类型 UNIQUE
```

【例 3.13】定义数据表 tb_dept2，指定部门的名称唯一，SQL 语句如下：

```
CREATE TABLE tb_dept2  
(  
    id      NUMBER(11) PRIMARY KEY,  
    name    VARCHAR2(22) UNIQUE,  
    location VARCHAR2(50)  
);
```

（2）在定义完所有列之后指定唯一性约束，语法规则如下：



```
[CONSTRAINT <约束名>] UNIQUE(<字段名>)
```

【例 3.14】定义数据表 tb_dept3，指定部门的名称唯一，SQL 语句如下：

```
CREATE TABLE tb_dept3
(
  id      NUMBER(11) PRIMARY KEY,
  name    VARCHAR2(22),
  location VARCHAR2(50),
  CONSTRAINT STH UNIQUE(name)
);
```

UNIQUE 和 PRIMARY KEY 的区别：一个表中可以有多个字段声明为 UNIQUE，但只能有一个 PRIMARY KEY 声明；声明为 PRIMARY KEY 的列不允许有空值，但是声明为 UNIQUE 的字段允许空值（NULL）的存在。

2. 在修改表时添加唯一性约束

修改表时也可以添加唯一性约束，具体 SQL 语法格式如下：

```
ALTER TABLE 数据表名称
ADD CONSTRAINT 约束名称 UNIQUE ( 字段名称);
```

【例 3.15】将 tb_emp5 表上的字段 name 添加唯一性约束，SQL 语句如下：

```
ALTER TABLE tb_emp5
ADD CONSTRAINT unq_name UNIQUE (name);
```

语句执行完成后，即为 tb_emp5 表的 name 字段添加了唯一性约束。

3. 移除唯一性约束

对于不需要的唯一性约束，可以将其移除，具体的语法格式如下：

```
ALTER TABLE 数据表名称
DROP CONSTRAINTS 约束名称;
```

【例 3.16】移除数据表 tb_emp5 的唯一性约束，SQL 语句如下：

```
ALTER TABLE tb_emp5
DROP CONSTRAINTS unq_name;
```

上述语句执行完成后，即可成功移除唯一性约束。

3.3.6 使用默认约束

默认约束（Default Constraint）指定某列的默认值。如男性同学较多，性别就可以默认为“男”。如果插入一条新的记录时没有为这个字段赋值，那么系统会自动为这个字段赋值为“男”。



默认约束的语法规则如下：

字段名 数据类型 DEFAULT 默认值

【例 3.17】定义数据表 tb_emp7，指定员工的部门编号默认为 1111，SQL 语句如下：

```
CREATE TABLE tb_emp7
(
  id      NUMBER(11) PRIMARY KEY,
  name    VARCHAR2(25) NOT NULL,
  deptId  NUMBER(11) DEFAULT 1111,
  salary  NUMBER(9,2)
);
```

以上语句执行成功之后，表 tb_emp7 上的字段 deptId 拥有了一个默认的值 1111，新插入的记录如果没有指定部门编号，则默认都为 1111。

3.3.7 使用检查约束

检查约束为 CHECK 约束，规定每一列能够输入的值，从而可以确保数值的正确性。例如，性别字段中可以规定只能输入男或者女，此时可以用到检查约束。

1. 创建检查约束

检查约束的语法规则如下：

CONSTRAINT 检查约束名称 CHECK (检查条件)

【例 3.18】定义数据表 tb_emp8，指定员工的性别只能输入“男”或者“女”，SQL 语句如下：

```
CREATE TABLE tb_emp8
(
  id      NUMBER(11) PRIMARY KEY,
  name    VARCHAR2(25) NOT NULL,
  gender  VARCHAR2(2),
  age     NUMBER(2),
  CONSTRAINT CHK_GENDER CHECK (GENDER='男' or GENDER='女')
);
```

以上语句执行成功之后，表 tb_emp8 上的字段 gender 添加了检查约束，新插入记录时只能输入“男”或者“女”。

2. 在修改表时添加检查约束

修改表时也可以添加检查约束，具体 SQL 语法格式如下：



```
ALTER TABLE 数据表名称  
ADD CONSTRAINT 约束名称 CHECK ( 检查条件 );
```

【例 3.19】将 tb_emp8 表上的字段 age 添加检查约束，规定年龄输入值在 15~25 之间。SQL 语句如下：

```
ALTER TABLE tb_emp8  
ADD CONSTRAINT chk_age CHECK (age>=15 and age<=25);
```

语句执行完成后，即为 tb_emp8 表的 age 字段添加了检查约束。

3. 移除检查约束

对于不需要的检查约束，可以将其移除，具体的语法格式如下：

```
ALTER TABLE 数据表名称  
DROP CONSTRAINTS 约束名称;
```

【例 3.20】移除数据表 tb_emp8 的检查约束 chk_age，SQL 语句如下：

```
ALTER TABLE tb_emp8  
DROP CONSTRAINTS chk_age;
```

上述语句执行完成后，即可成功移除检查约束 chk_age。

3.3.8 设置表的属性值自动增加

在数据库应用中，经常希望在每次插入新记录时，系统自动生成字段的主键值。可以通过为表主键添加 GENERATED BY DEFAULT AS IDENTITY 关键字来实现。默认，在 Oracle 中自增值的初始值是 1，每新增一条记录，字段值自动加 1。一个表只能有一个字段使用自增约束，且该字段必须为主键的一部分。

设置自增约束的语法规则如下：

```
字段名 数据类型 GENERATED BY DEFAULT AS IDENTITY
```

【例 3.21】定义数据表 tb_emp9，指定员工的编号自动递增，SQL 语句如下：

```
CREATE TABLE tb_emp9  
(  
id      NUMBER(11) GENERATED BY DEFAULT AS IDENTITY  
name    VARCHAR2(25) NOT NULL,  
deptId  NUMBER(11),  
salary  NUMBER(9,2)  
);
```

上述例子执行后，会创建名称为 tb_emp9 的数据表。表 tb_emp9 中的 id 字段的值在添加



记录的时候会自动增加，在插入记录的时候，默认的自增字段 id 的值从 1 开始，每次添加一条新记录，该值自动加 1。

例如，执行如下插入语句：

```
SQL> INSERT INTO tb_emp9 (name)
VALUES ('张三');
SQL> INSERT INTO tb_emp9 (name);
VALUES ('程普');
```

语句执行完后，tb_emp9 表中增加 2 条记录，在这里并没有输入 id 的值，但系统已经自动添加该值，使用 SELECT 命令查看记录，如下所示。

```
SQL> SELECT * FROM tb_emp9;

   ID NAME      DEPTID      SALARY
-----
    1 张三
    2 程普
```

提示

这里使用 INSERT 声明向表中插入记录的方法，一次只能插入一行数据。如果想一次插入多行数据，需要使用 INSERT INTO ... SELECT ... 子查询的方式。具体使用方法参考本章后面的章节。

3.4 查看数据表结构

使用 SQL 语句创建好数据表之后，可以查看表结构的定义，以确认表的定义是否正确。在 Oracle 中，查看表结构可以使用 DESCRIBE 语句。

DESCRIBE/DESC 语句可以查看表的字段信息，其中包括：字段名、字段数据类型、是否为主键、是否有默认值等。语法规则如下：

```
DESCRIBE 表名;
```

或者简写为：

```
DESC 表名;
```

【例 3.22】 分别使用 DESCRIBE 和 DESC 查看表 tb_dept1 和表 tb_emp1 的表结构。查看 tb_dept1 表结构，SQL 语句如下：

```
SQL> DESCRIBE tb_dept1;

名称      空值      类型
-----
ID        NOT NULL  NUMBER(11)
```



```
NAME      NOT NULL VARCHAR2(22)
LOCATION    VARCHAR2(50)
```

查看 tb_emp1 表结构，SQL 语句如下：

```
SQL> DESC tb_emp1;
名称      空值 类型
-----
ID         NUMBER(11)
NAME       VARCHAR2(25)
DEPTID     NUMBER(11)
SALARY     NUMBER(9,2)
```

3.5 修改数据表

修改表指的是修改数据库中已经存在的数据表的结构。Oracle 使用 ALTER TABLE 语句修改表。常用的修改表的操作有：修改表名，修改字段数据类型或字段名，增加和删除字段，修改字段的排列位置，更改表的存储引擎，删除表的外键约束等。本节将对和修改表有关的操作进行讲解。

3.5.1 修改表名

Oracle 是通过 ALTER TABLE 语句来实现表名的修改的，具体的语法规则如下：

```
ALTER TABLE <旧表名> RENAME TO <新表名>;
```

【例 3.23】将数据表 tb_dept3 改名为 tb_deptment3。

执行修改表名操作之前，使用 PESC 查看数据表 tb_dept3。

```
SQL> DESC tb_dept3;
名称      空值      类型
-----
ID         NOT NULL NUMBER(11)
NAME              VARCHAR2(22)
LOCATION        VARCHAR2(50)
```

使用 ALTER TABLE 将表 tb_dept3 改名为 tb_deptment3，SQL 语句如下：

```
ALTER TABLE tb_dept3 RENAME TO tb_deptment3;
```

语句执行之后，检验表 tb_dept3 是否改名成功。

使用 DESC 查看数据表 tb_dept3 是否还存在，结果如下：



Oracle 12c 从零开始学（视频教学版）

```
SQL> DESC tb_dept3;
ERROR:
ORA-04043: 对象 tb_dept3 不存在
```

使用 DESC 查看数据表 tb_deptment3，结果如下：

```
SQL> DESC tb_deptment3;
名称      空值      类型
-----
ID          NOT NULL  NUMBER(11)
NAME                VARCHAR2(22)
LOCATION          VARCHAR2(50)
```

经过比较可以看到，tb_dept3 表已经改名为 tb_deptment3 表。

3.5.2 修改字段的数据类型

修改字段的数据类型，就是把字段的数据类型转换成另一种数据类型。在 Oracle 中修改字段数据类型的语法规则如下：

```
ALTER TABLE <表名> MODIFY <字段名> <数据类型>
```

其中，“表名”指要修改数据类型的字段所在表的名称，“字段名”指需要修改的字段，“数据类型”指修改后字段的新数据类型。

【例 3.24】将数据表 tb_dept1 中 name 字段的数据类型由 VARCHAR2(22)修改成 VARCHAR2(30)。

执行修改字段的数据类型操作之前，使用 DESC 查看 tb_dept1 表结构，结果如下：

```
SQL> DESC tb_dept1;
名称      空值      类型
-----
ID          NOT NULL  NUMBER(11)
NAME  NOT NULL  VARCHAR2(22)
LOCATION          VARCHAR2(50)
```

可以看到现在 name 字段的数据类型为 VARCHAR2(22)，下面修改其类型。输入如下 SQL 语句并执行：

```
ALTER TABLE tb_dept1 MODIFY name VARCHAR2(30);
```

再次使用 DESC 查看表，结果如下：

```
SQL> DESC tb_dept1;
名称      空值      类型
-----
```



```
ID          NOT NULL  NUMBER(11)
NAME        NOT NULL  VARCHAR2(30)
LOCATION      VARCHAR2(50)
```

语句执行之后，检验会发现表 `tb_dept1` 中 `name` 字段的数据类型已经修改成了 `VARCHAR2(30)`，修改成功。

3.5.3 修改字段名

Oracle 中修改字段名的语法规则如下：

```
ALTER TABLE <表名> RENAME COLUMN <旧字段名> TO<新字段名> ;
```

其中，“旧字段名”指修改前的字段名，“新字段名”指修改后的字段名，“新数据类型”指修改后的数据类型，如果不需要修改字段的数据类型，可以将新数据类型设置成与原来一样，但数据类型不能为空。

【例 3.25】将数据表 `tb_dept1` 中的 `location` 字段名称改为 `loc`，数据类型保持不变，SQL 语句如下：

```
ALTER TABLE tb_dept1 RENAME COLUMN location TO loc ;
```

使用 `DESC` 查看表 `tb_dept1`，会发现字段的名称已经修改成功，结果如下：

```
SQL> DESC tb_dept1;
名称  空值      类型
----  -
ID    NOT NULL  NUMBER(11)
NAME  NOT NULL  VARCHAR2(30)
LOC             VARCHAR2(50)
```

提示

由于不同类型的数据在计算机中存储的方式及长度并不相同，修改数据类型可能会影响到数据表中已有的数据记录。因此，当数据表中已经有数据时，不要轻易修改数据类型。

3.5.4 添加字段

随着业务需求的变化，可能需要在已经存在的表中添加新的字段。一个完整字段包括字段名、数据类型、完整性约束。添加字段的语法格式如下：

```
ALTER TABLE <表名> ADD <新字段名> <数据类型>
```

其中，“新字段名”为需要添加的字段名称。

1. 添加无完整性约束条件的字段

【例 3.26】在数据表 `tb_dept1` 中添加一个没有完整性约束的 `NUMBER` 类型的字段 `managerId`（部门经理编号），SQL 语句如下：



Oracle 12c 从零开始学（视频教学版）

```
ALTER TABLE tb_dept1 ADD managerId NUMBER(10);
```

使用 DESC 查看表 tb_dept1，会发现在表的最后添加了一个名为 managerId 的 NUMBER 类型的字段，结果如下：

```
SQL> DESC tb_dept1;
 名称      空值      类型
-----
ID          NOT NULL  NUMBER(11)
NAME        NOT NULL  VARCHAR2(30)
LOC                      VARCHAR2(50)
MANAGERID   NUMBER(10)
```

2. 添加有完整性约束条件的字段

【例 3.27】在数据表 tb_dept1 中添加一个不能为空的 VARCHAR2(12)类型的字段 column1, SQL 语句如下：

```
ALTER TABLE tb_dept1 ADD column1 VARCHAR2(12) not null;
```

使用 DESC 查看表 tb_dept1，会发现在表的最后添加了一个名为 column1 的 VARCHAR2(12)类型且不为空的字段，结果如下：

```
SQL> DESC tb_dept1;
 名称      空值      类型
-----
ID          NOT NULL  NUMBER(11)
NAME        NOT NULL  VARCHAR2(30)
LOC                      VARCHAR2(50)
MANAGERID   NUMBER(10)
COLUMN1     NOT NULL  VARCHAR2(12)
```

3.5.5 删除字段

删除字段是将数据表中的某个字段从表中移除，语法格式如下：

```
ALTER TABLE <表名> DROP COLUMN <字段名>;
```

其中，“字段名”指需要从表中删除的字段名称。

【例 3.28】删除数据表 tb_dept1 中的 column1 字段。

首先，执行删除字段操作之前，使用 DESC 查看 tb_dept1 表结构，结果如下：

```
SQL> DESC tb_dept1;
 名称      空值      类型
-----
```



```
ID          NOT NULL NUMBER(11)
NAME        NOT NULL VARCHAR2(30)
LOC          VARCHAR2(50)
MANAGERID    NUMBER(10)
COLUMN1     NOT NULL VARCHAR2(12)
```

删除 column1 字段，SQL 语句如下：

```
ALTER TABLE tb_dept1 DROP COLUMN column1;
```

再次使用 DESC 查看表 tb_dept1，结果如下：

```
SQL> DESC tb_dept1;
名称          空值          类型
-----
ID            NOT NULL    NUMBER(11)
NAME          NOT NULL    VARCHAR2(30)
LOC            VARCHAR2(50)
MANAGERID      NUMBER(10)
```

可以看到，tb_dept1 表中已经不存在名称为 column1 的字段，删除字段成功。

提示

在删除表中的列时，常常在列后添加 CASCADE CONSTRAINTS，目的是将与该列相关的约束一并删除掉。

3.6 删除数据表

删除数据表就是将数据库中已经存在的表从数据库中删除。注意，在删除表的同时，表的定义和表中所有的数据均会被删除。因此，在进行删除操作前，最好对表中的数据做个备份，以免造成无法挽回的后果。本节将详细讲解数据表的删除方法。

3.6.1 删除没有被关联的表

在 Oracle 中，使用 DROP TABLE 可以一次删除一个或多个没有被其他表关联的数据表。语法格式如下：

```
DROP TABLE 表名;
```

在前面的例子中，已经创建了名为 tb_dept2 的数据表。如果没有，读者可输入语句，创建该表。下面使用删除语句将该表删除。

【例 3.29】删除数据表 tb_dept2，SQL 语句如下：



```
DROP TABLE tb_dept2;
```

语句执行完毕之后，使用 DESC 语句查看当前数据库中所有的表，SQL 语句如下：

```
SQL> DESC tb_dept2;
ERROR:
-----
错误：对象 TB_DEPT2 不存在
```

从执行结果可以看到，数据表列表中已经不存在名称为 tb_dept2 的表，删除操作成功。

3.6.2 删除被其他表关联的主表

数据表之间存在外键关联的情况下，如果直接删除父表，结果会显示失败。原因是直接删除，将破坏表的参照完整性。如果必须要删除，可以先删除与它关联的子表，再删除父表，只是这样同时删除了两个表中的数据。但有的情况下可能要保留子表，这时如要单独删除父表，只需将关联的表的外键约束条件取消，然后就可以删除父表，下面讲解这种方法。

在数据库中创建两个关联表，首先创建表 tb_dept2，SQL 语句如下：

```
CREATE TABLE tb_dept2
(
  id      NUMBER(11) PRIMARY KEY,
  name    VARCHAR2(22),
  location VARCHAR2(50)
);
```

接下来创建表 tb_emp，SQL 语句如下：

```
CREATE TABLE tb_emp
(
  id      NUMBER(11) PRIMARY KEY,
  name    VARCHAR2(25),
  deptId  NUMBER(11),
  salary  NUMBER(9,2),
  CONSTRAINT fk_emp_dept FOREIGN KEY (deptId) REFERENCES tb_dept2(id)
);
```

可以看到，以上执行结果创建了两个关联表 tb_dept2 和 tb_emp，其中 tb_emp 表为子表，具有名称为 fk_emp_dept 的外键约束，tb_dept2 为父表，其主键 id 被子表 tb_emp 所关联。

【例 3.30】删除被数据表 tb_emp 关联的数据表 tb_dept2。

首先直接删除父表 tb_dept2，输入删除语句如下：

```
SQL> DROP TABLE tb_dept2;
错误报告：
```

SQL 错误：ORA-02449：表中的唯一主键被外键引用

可以看到，如前所述，在存在外键约束时，主表不能被直接删除。

接下来，移除 tb_emp 外键约束，SQL 语句如下：

```
ALTER TABLE tb_emp DROP CONSTRAINTS fk_emp_dept;
```

语句成功执行后，将取消表 tb_emp 和表 tb_dept2 之间的关联关系。此时，可以输入删除语句，将原来的父表 tb_dept2 删除，SQL 语句如下：

```
DROP TABLE tb_dept2;
```

最后通过 DESC 语句查看数据表列表，如下所示：

```
SQL> DESC tb_dept2;
```

```
ERROR:
```

```
-----
```

```
错误：对象 TB_DEPT2 不存在
```

可以看到，数据表列表中已经不存在名称为 tb_dept2 的表。

3.7 综合案例——数据表的基本操作

本章全面介绍了 Oracle 中数据表的各种操作，如创建表、添加各类约束、查看表结构，以及修改和删除表。读者应该掌握这些基本的操作，为以后的学习打下坚实的基础。在这里，给出一个综合案例，让读者全面回顾一下本章的知识要点，并通过这些操作来检验自己是否已经掌握了数据表的常用操作。

1. 案例目的

创建、修改和删除表，掌握数据表的基本操作。

按照表 3-3 和表 3-4 给出的表结构在 company 数据库中创建两个数据表 offices 和 employees，按照操作过程完成对数据表的基本操作。

表 3-3 offices 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
officeCode	NUMBER(10)	是	否	是	是	否
city	VARCHAR2(50)	否	否	是	否	否
address	VARCHAR2(50)	否	否	否	否	否
country	VARCHAR2(50)	否	否	是	否	否
postalCode	VARCHAR2(15)	否	否	否	是	否

表 3-4 employees 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
employeeNumber	NUMBER(11)	否	否	是	是	是
lastName	VARCHAR2(50)	否	否	是	否	否
firstName	VARCHAR2(50)	否	否	是	否	否
mobile	VARCHAR2(25)	否	否	否	是	否
officeCode	NUMBER(10)	否	是	是	否	否
jobTitle	VARCHAR2(50)	否	否	是	否	否
birth	DATE	否	否	是	否	否
note	VARCHAR2(255)	否	否	否	否	否
sex	VARCHAR2(5)	否	否	否	否	否

2. 案例操作过程

步骤 01 创建表 offices。

创建表 offices 的语句如下：

```
CREATE TABLE offices
(
  officeCode NUMBER(10) NOT NULL,
  city        VARCHAR2(50) NOT NULL,
  address     VARCHAR2(50),
  country     VARCHAR2(50) NOT NULL,
  postalCode  VARCHAR2(15),
  PRIMARY KEY (officeCode)
);
```

执行成功之后，使用 DESC 语句查看数据表 offices，语句如下：

```
SQL> DESC offices;
名称          空值      类型
-----
OFFICECODE NOT NULL NUMBER(10)
CITY        NOT NULL VARCHAR2(50)
ADDRESS     VARCHAR2(50)
COUNTRY     NOT NULL VARCHAR2(50)
POSTALCODE  VARCHAR2(15)
```

可以看到，数据库中已经有了数据表 offices，创建成功。

步骤 02 创建表 employees。

创建表 employees 的语句如下：

```
CREATE TABLE employees
(
  employeeNumber NUMBER(11) GENERATED BY DEFAULT AS IDENTITY,
  lastName        VARCHAR2(50) NOT NULL,
  firstName       VARCHAR2(50) NOT NULL,
  mobile          VARCHAR2(25) ,
  officeCode      NUMBER(10) NOT NULL,
  jobTitle        VARCHAR2(50) NOT NULL,
  birth           DATE NOT NULL,
  note            VARCHAR2(255),
  sex             VARCHAR2(5),
  CONSTRAINT office_fk FOREIGN KEY(officeCode) REFERENCES offices(officeCode)
);
```

执行成功之后，使用 DESC 语句查看数据表 employees，语句如下：

```
SQL>DESC employees;
名称          空值      类型
-----
EMPLOYEEENUMBER NOT NULL NUMBER(11)
LASTNAME        NOT NULL VARCHAR2(50)
FIRSTNAME       NOT NULL VARCHAR2(50)
MOBILE          VARCHAR2(25)
OFFICECODE      NOT NULL NUMBER(10)
JOBTITLE        NOT NULL VARCHAR2(50)
BIRTH           NOT NULL      DATE
NOTE            VARCHAR2(255)
SEX             VARCHAR2(5)
```

可以看到，现在数据库中已经创建好了 employees 数据表。

步骤 03 将表 employees 的 birth 字段改名为 employee_birth。

修改字段名，需要用到 ALTER TABLE 语句，输入语句如下：

```
ALTER TABLE employees RENAME COLUMN birth TO employee_birth ;
table EMPLOYEES 已变更。
```

结果显示执行成功，使用 DESC 查看修改后的结果如下：

```
SQL>DESC employees;
```


名称	空值	类型
EMPLOYEEENUMBER	NOT NULL	NUMBER(11)
LASTNAME	NOT NULL	VARCHAR2(50)
FIRSTNAME	NOT NULL	VARCHAR2(50)
MOBILE		VARCHAR2(25)
OFFICECODE	NOT NULL	NUMBER(10)
JOBTITLE	NOT NULL	VARCHAR2(50)
EMPLOYEE_BIRTH	NOT NULL	DATE
NOTE		VARCHAR2(255)
SEX		VARCHAR2(5)

可以看到，表中只有 `employee_birth` 字段，已经没有名称为 `birth` 的字段了，修改名称成功。

步骤 04 修改 `sex` 字段，数据类型为 `VARCHAR2(2)`，非空约束。

修改字段数据类型，需要用到 `ALTER TABLE` 语句，输入语句如下：

```
SQL>ALTER TABLE employees MODIFY sex VARCHAR2(2) NOT NULL;
```

结果显示执行成功，使用 `DESC` 查看修改后的结果如下：

```
SQL>DESC employees;
名称          空值      类型
-----
EMPLOYEEENUMBER NOT NULL  NUMBER(11)
LASTNAME        NOT NULL  VARCHAR2(50)
FIRSTNAME       NOT NULL  VARCHAR2(50)
MOBILE          VARCHAR2(25)
OFFICECODE      NOT NULL  NUMBER(10)
JOBTITLE        NOT NULL  VARCHAR2(50)
EMPLOYEE_BIRTH  NOT NULL  DATE
NOTE            VARCHAR2(255)
SEX             NOT NULL  VARCHAR2(2)
```

从执行结果可以看到，`sex` 字段的数据类型由前面的 `VARCHAR2(5)` 修改为 `VARCHAR2(2)`，且其空值列显示为 `NOT NULL`，表示该列不允许空值，修改成功。

步骤 05 删除字段 `note`。

删除字段，需要用到 `ALTER TABLE` 语句，输入语句如下：


```
SQL> ALTER TABLE employees DROP COLUMN note;
```

语句执行成功后, 使用 `DESC employees;` 查看语句执行后的结果:

```
SQL> DESC employees;
```

名称	空值	类型
EMPLOYEEID	NOT NULL	NUMBER(11)
LASTNAME	NOT NULL	VARCHAR2(50)
FIRSTNAME	NOT NULL	VARCHAR2(50)
MOBILE		VARCHAR2(25)
OFFICECODE	NOT NULL	NUMBER(10)
JOBTITLE	NOT NULL	VARCHAR2(50)
EMPLOYEE_BIRTH	NOT NULL	DATE
SEX	NOT NULL	VARCHAR2(2)

可以看到, `DESC` 语句返回了 8 个列字段, `note` 字段已经不在表结构中, 删除字段成功。

步骤 06 增加字段名 `favoriate_activity`, 数据类型为 `VARCHAR2(100)`。

增加字段, 需要用到 `ALTER TABLE` 语句, 输入语句如下:

```
SQL> ALTER TABLE employees ADD favoriate_activity VARCHAR2(100);
table EMPLOYEES 已变更。
```

结果显示执行语句成功, 使用 `DESC employees;` 查看语句执行后的结果:

```
SQL> DESC employees;
```

名称	空值	类型
EMPLOYEEID	NOT NULL	NUMBER(11)
LASTNAME	NOT NULL	VARCHAR2(50)
FIRSTNAME	NOT NULL	VARCHAR2(50)
MOBILE		VARCHAR2(25)
OFFICECODE	NOT NULL	NUMBER(10)
JOBTITLE	NOT NULL	VARCHAR2(50)
EMPLOYEE_BIRTH	NOT NULL	DATE
SEX	NOT NULL	VARCHAR2(2)
FAVORIATE_ACTIVITY		VARCHAR2(100)

可以看到, 数据表 `employees` 中增加了一个新的列 `favoriate_activity`, 数据类型为 `VARCHAR2(100)`, 允许空值, 添加新字段成功。

步骤 07 删除表 offices。

在创建表 employees 时，设置了表的外键，该表关联了其父表的 officeCode 主键。如前面所述，删除关联表时，要先删除子表 employees 的外键约束，才能删除父表。因此，必须先删除 employees 表的外键约束。

1) 删除 employees 表的外键约束，输入如下语句：

```
SQL>ALTER TABLE employees DROP CONSTRAINTS office_fk;
table EMPLOYEES 已变更。
```

其中，office_fk 为 employees 表的外键约束的名称，即创建外键约束时 CONSTRAINT 关键字后面的参数，结果显示语句执行成功，现在可以删除 offices 父表。

2) 删除表 offices，输入如下语句：

```
SQL>DROP TABLE offices;
table OFFICES 已删除。
```

结果显示执行删除操作成功，使用 DESC 语句查看数据库中的表，结果如下：

```
SQL> DESC offices;
ERROR:
-----
错误：对象 OFFICES 不存在
```

可以看到，数据库中已经没有名称为 offices 的表了，删除表成功。

步骤 08 将表 employees 名称修改为 employees_info。

修改数据表名，需要用到 ALTER TABLE 语句，输入语句如下：

```
SQL>ALTER TABLE employees RENAME TO employees_info;
table EMPLOYEES 已变更。
```

结果显示执行语句成功，使用 DESC 语句查看执行结果：

```
SQL>DESC employees_info;

名称                空值      类型
-----
EMPLOYEEENUMBER    NOT NULL  NUMBER(11)
LASTNAME           NOT NULL  VARCHAR2(50)
FIRSTNAME          NOT NULL  VARCHAR2(50)
MOBILE              VARCHAR2(25)
OFFICECODE         NOT NULL  NUMBER(10)
```

JOBTITLE	NOT NULL	VARCHAR2 (50)
EMPLOYEE_BIRTH	NOT NULL	DATE
SEX	NOT NULL	VARCHAR2 (2)
FAVORITE_ACTIVITY		VARCHAR2 (100)

3.8 疑难解惑

疑问 1：SQL Plus 中无法进行复制和粘贴操作怎么办？

解答：在 SQL Plus 主界面中右击，在弹出的快捷菜单中选择【属性】命令，如图 3-14 所示。

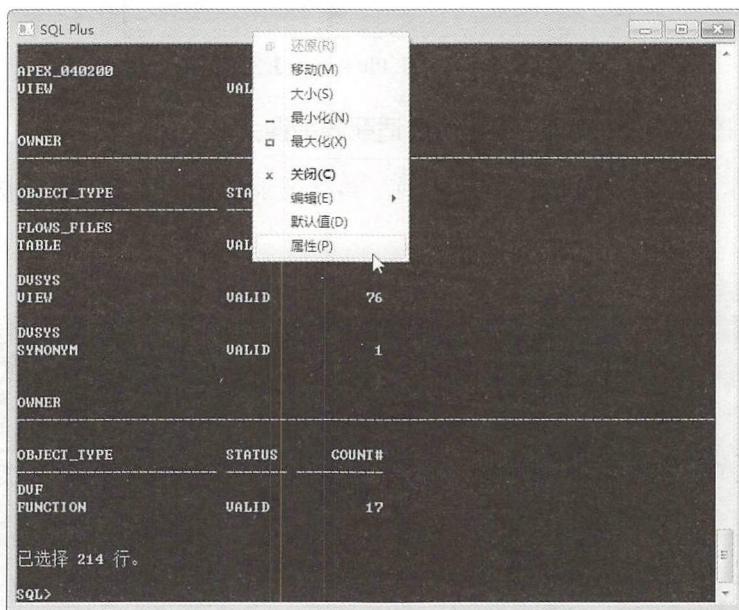


图 3-14 选择【属性】命令

在打开的对话框中选择【选项】选项卡，然后选中【快速编辑模式】复选框，单击【确定】按钮即可，如图 3-15 所示。

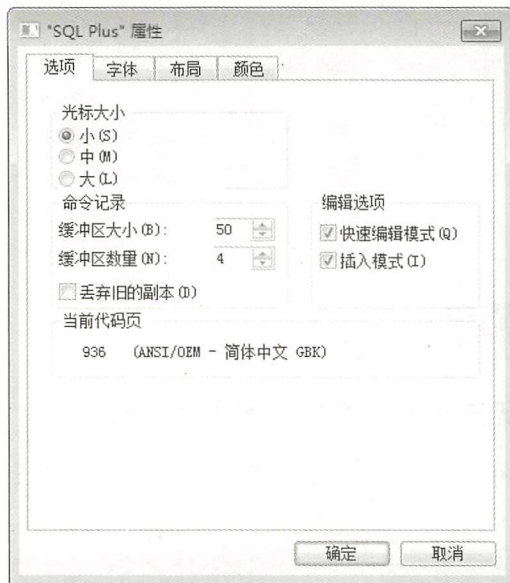


图 3-15 【SQL Plus 属性】对话框

疑问 2：无法登录 SQL Plus，提示协议适配器错误。

新创建数据库后，然后删除新建的数据库，再次登录 SQL Plus 时，提示协议适配器错误，如图 3-16 所示。

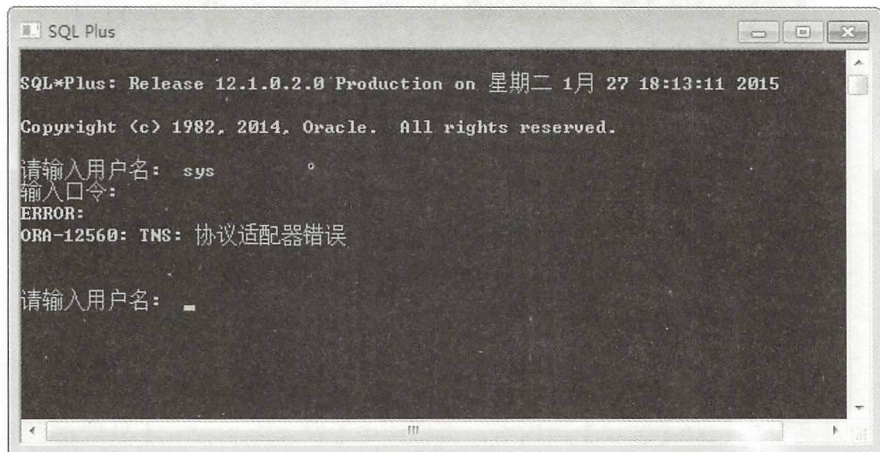


图 3-16 错误提示信息

解答：删除数据库后，ORACLE_SID 并没有修改过来，此时需要恢复值为“orcl”。具体操作方法为：在【运行】对话框中输入“regedit”并按【Enter】键，打开系统的注册表页面，依次找到 HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\KEY_OraDB12Home1 项目下的 ORACLE_SID，右击并在弹出的快捷菜单中选择【修改】命令，打开【编辑字符串】对话框，修改数据为“orcl”，单击【确定】按钮即可，如图 3-17 所示。

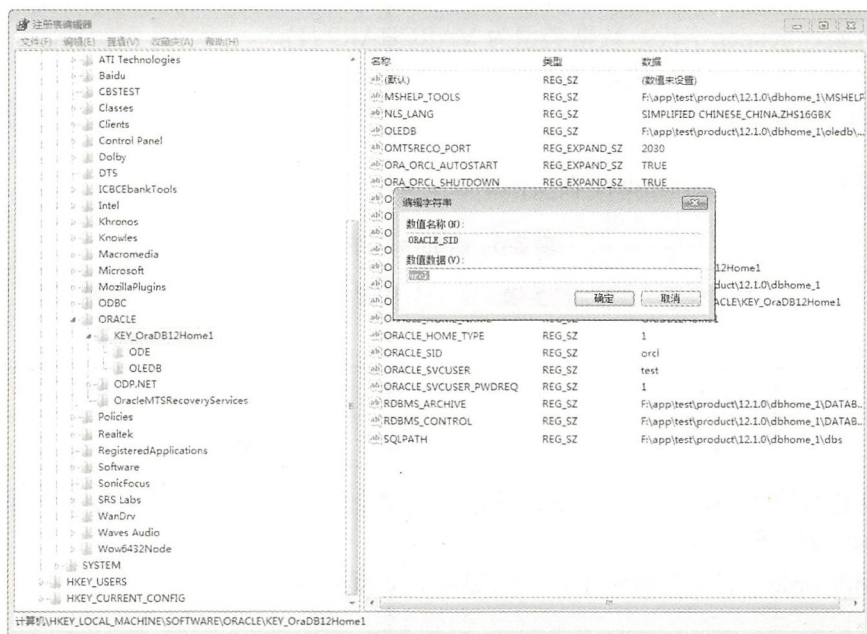


图 3-17 修改 ORACLE_SID 的值

疑问 3：每一个表中都要有一个主键吗？

解答：并不是每一个表中都需要主键，一般的，如果多个表之间进行连接操作时，需要用到主键。因此并不需要为每个表建立主键，而且有些情况最好不使用主键。

3.9 经典习题

1. 创建数据表 customers，customers 表结构如表 3-5 所示，按要求进行操作。

表 3-5 customers 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
c_num	NUMBER(11)	是	否	是	是	是
c_name	VARCHAR2(50)	否	否	否	否	否
c_contact	VARCHAR2(50)	否	否	否	否	否
c_city	VARCHAR2(50)	否	否	否	否	否
c_birth	DATE	否	否	是	否	否

(1) 创建数据表 customers，在 c_num 字段上添加主键约束和自增约束，在 c_birth 字段上添加非空约束。

(2) 将 c_name 字段数据类型改为 VARCHAR2(70)。

(3) 将 c_contact 字段改名为 c_phone。

(4) 增加 c_gender 字段，数据类型为 VARCHAR2(1)。

(5) 将表名修改为 customers_info。

(6) 删除字段 c_city。

2. 创建数据表 orders，orders 表结构如表 3-6 所示，按要求进行操作。

表 3-6 orders 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
o_num	NUMBER(11)	是	否	是	是	是
o_date	DATE	否	否	否	否	否
c_id	VARCHAR2(50)	否	是	否	否	否

(1) 创建数据表 orders，在 o_num 字段上添加自增约束，在 c_id 字段上添加外键约束，关联 customers 表中的主键 c_num。

(2) 删除 orders 表的外键约束，然后删除表 customers。

第 4 章

◀ 数据类型和运算符 ▶

数据表由多列字段构成，每一个字段指定了不同的数据类型。指定字段的数据类型之后，也就决定了向字段插入的数据内容，例如，当要插入数值的时候，可以将它们存储为整数类型，也可以将它们存储为字符串类型；不同的数据类型也决定了 Oracle 在存储它们的时候使用的方式，以及在使用它们的时候选择什么运算符进行运算。本章将介绍 Oracle 中的数据类型和常见的运算符。

- 熟悉常见数据类型的概念和区别
- 掌握如何选择数据类型
- 熟悉常见运算符的概念和区别

4.1 Oracle 数据类型介绍

Oracle 支持多种数据类型，主要有数值类型、日期/时间类型和字符串类型等。

- (1) 数值类型：包括整数类型和小数类型。
- (2) 日期/时间类型：包括 DATE 和 TIMESTAMP。
- (3) 字符串类型：包括 CHAR、VARCHAR2、NVARCHAR2、NCHAR 和 LONG 五种。

4.1.1 数值类型

数值型数据类型主要用来存储数字，Oracle 提供了多种数值数据类型，不同的数据类型提供不同的取值范围，可以存储的值范围越大，其所需要的存储空间也会越大。Oracle 的数值类型主要通过 `number(m,n)` 语句来实现。使用的语法格式如下：

```
number(m,n)
```

其中，`m` 的取值范围为 1~38，`n` 的取值范围为 -84~127。

`number(m,n)` 是可变长的数值列，允许 0、正值及负值，`m` 是所有有效数字的位数，`n` 是小数点以后的位数。如：

```
number(5,2)
```


则这个字段的最大值是 999.99，如果数值超出了位数限制就会被截取多余的位数。
如：

```
number(5,2)
```

但在一行数据中的这个字段输入 575.316，则真正保存到字段中的数值是 575.32。
如：

```
number(3,0)
```

输入 575.316，真正保存的数据是 575。对于整数，可以省略后面的 0，直接表示如下：

```
number(3)
```

在第 3 章“创建数据表”章节中，有如下创建表的语句：

```
CREATE TABLE tb_emp1
(
  id      NUMBER(11),
  name    VARCHAR2(25),
  deptId  NUMBER(11),
  salary  NUMBER(9,2)
);
```

id 字段的数据类型为 NUMBER(11)，注意到后面的数字 11，这表示的是该数据类型指定的最大长度，如果插入数值的位数大于 11，则会弹出以下错误信息：

错误报告：

SQL 错误：ORA-01438：值大于为此列指定的允许精度

【例 4.1】创建表 tmp1，其中字段 x、y 数据类型依次为 NUMBER(4)、NUMBER(6)，SQL 语句如下：

```
CREATE TABLE tmp1 ( x NUMBER(4), y NUMBER(6));
```

执行成功之后，使用 DESC 查看表结构，结果如下：

```
SQL> DESC tmp1;
名称 空值 类型
--- --
X      NUMBER(4)
Y      NUMBER(6)
```

不同的数值类型的长度不同，取值范围不同，并且需要不同的存储空间，因此，应该根据实际需要选择最合适的类型，这样有利于提高查询的效率和节省存储空间。

【例 4.2】创建表 tmp2，其中字段 x、y、z 数据类型依次为 NUMBER(5,1)、NUMBER(5,3) 和 NUMBER(5,2)，向表中插入数据 5.12、5.15 和 5.123，SQL 语句如下：

```
CREATE TABLE tmp2 ( x NUMBER (5,1), y NUMBER (5,3), z NUMBER (5,2) );
```

向表中插入数据:

```
SQL>INSERT INTO tmp2 VALUES(5.12, 5.15, 5.123);
```

插入数据后, 查询输入的数据信息。SQL 语句如下:

```
SQL> SELECT * FROM tmp2;
```

```

      X          Y          Z
-----
      5.1        5.15        5.12

```

从结果可以看出, 5.12 和 5.123 分别被存储为 5.1 和 5.12。

4.1.2 日期与时间类型

Oracle 中表示日期的数据类型, 主要包括 DATE 和 TIMESTAMP。具体含义和区别如表 4-1 所示。

表 4-1 日期与时间数据类型

类型名称	说明
DATE	用来存储日期和时间, 取值范围是公元前 4712 年 1 月 1 日到公元 9999 年 12 月 31 日
TIMESTAMP	用来存储日期和时间, 与 DATE 类型的区别就是显示日期和时间时更精确, DATE 类型的时间精确到秒, 而 TIMESTAMP 类型可以精确到小数秒, TIMESTAMP 存放日期和时间还能显示上午、下午和时区

【例 4.3】创建数据表 tmp3, 定义数据类型为 date 的字段 d, 向表中插入值'12-5 月-2010', SQL 语句如下:

首先创建表 tmp3:

```
CREATE TABLE tmp3( d date );
```

在插入数据之前, 需要知道数据库默认的时间格式, 查询 SQL 语句如下:

```
SQL> select sysdate from dual;
```

```

SYSDATE
-----
30-1月-15

```

向表中插入数据:

```
SQL> INSERT INTO tmp3 values('12-5月-2010');
```

查看结果如下:


```
SQL> SELECT * FROM tmp3;

D
-----
12-5月-10
```

如果用户想按照指定的格式输入时间，需要修改时间的默认格式。例如输入格式为年-月-日，修改的 SQL 语句如下。

```
SQL> alter session set nls_date_format='yyyy-mm-dd';
```

【例 4.4】创建数据表 tmp4，定义数据类型为 DATE 的字段 d，向表中插入“YYYY-MM-DD”和“YYYYMMDD”字符串格式日期，SQL 语句如下：

首先创建表 tmp4：

```
SQL> CREATE TABLE tmp4(d DATE);
```

修改日期的默认格式，SQL 语句如下：

```
SQL> alter session set nls_date_format='yyyy-mm-dd';
```

向表中插入“YYYY-MM-DD”格式日期：

```
SQL> INSERT INTO tmp4 values('1998-08-08');
```

向表中插入“YYYYMMDD”格式日期：

```
SQL> INSERT INTO tmp4 values('19980808');
```

查看插入结果：

```
SQL> SELECT * FROM tmp4;

D
-----
1998-08-08
1998-08-08
```

可以看到，各个不同类型的日期值都正确地插入到了数据表中。

【例 4.5】向 tmp4 表中插入系统当前日期，SQL 语句如下：

首先删除表中的数据：

```
DELETE FROM tmp4;
```

向表中插入系统当前日期：

```
SQL> INSERT INTO tmp4 values(SYSDATE );
```

查看插入结果:

```
SQL> SELECT * FROM tmp4;
```

```
D
```

```
-----
```

```
2015-01-30
```

【例 4.6】向 tmp4 表中插入系统日期和时间并指定格式, SQL 语句如下:
首先删除表中的数据:

```
DELETE FROM tmp4;
```

向表中插入系统当前日期:

```
SQL> INSERT INTO tmp4 values(to_date('2005-01-01 13:14:20','yyyy-MM-dd  
HH24:mi:ss'));
```

查看插入结果:

```
SQL> SELECT * FROM tmp4;
```

```
D
```

```
-----
```

```
2005-01-01
```

从结果可以看出, 只显示日期, 时间被省略掉了。

【例 4.7】创建数据表 tmp5, 定义数据类型为 TIMESTAMP 的字段 ts, 向表中插入值
'2013-9-16 17:03:00.9999', SQL 语句如下:

创建数据表 tmp5:

```
CREATE TABLE tmp5( ts TIMESTAMP);
```

向表中插入数据:

```
INSERT INTO tmp5 values (to_timestamp('2013-9-16 17:03:00.9999','yyyy-mm-dd  
hh24:mi:ss:ff'));
```

查看插入结果:

```
SQL>SELECT * FROM tmp5;
```

```
TS
```

```
-----
```

```
16-9月-13 05.03.00.999900000 下午
```


4.1.3 字符串类型

字符串类型用来存储字符串数据。Oracle 中的字符串类型指 CHAR、VARCHAR2、NCHAR、NVARCHAR2 和 LONG。表 4-2 列出了 Oracle 中的字符串数据类型。

表 4-2 Oracle 中的字符串数据类型

类型名称	说明	取值范围（字节）
CHAR	用于描述定长的字符型数据	0~2000
NCHAR	用来存储 Unicode 字符集的定长字符型数据	0~1000
VARCHAR2	用于描述可变长的字符型数据	0~4000
NVARCHAR2	用来存储 Unicode 字符集的可变长字符型数据	0~1000
LONG	用来存储变长的字符串	0~2GB

VARCHAR2、NVARCHAR2 和 LONG 类型是变长类型，对于其存储需求取决于列值的实际长度，而不是取决于类型的最大可能尺寸。例如，一个 VARCHAR2(10)列能保存最大长度为 10 个字符的一个字符串，实际的存储需要是字符串的长度。

【例 4.8】创建 tmp6 表，定义字段 ch 和 vch 数据类型依次为 CHAR(4)、VARCHAR2(4)，向表中插入数据“ab”，SQL 语句如下：

创建表 tmp6：

```
CREATE TABLE tmp6(  
  ch CHAR(4), vch VARCHAR2(4)  
);
```

插入数据：

```
INSERT INTO tmp6 VALUES('ab', 'ab');
```

查询 ch 字段的存储长度，执行 SQL 语句如下：

```
SQL> Select length(ch) from tmp6  
  
LENGTH (CH)  
-----  
4
```

查询 vch 字段的存储长度，执行 SQL 语句如下：

```
SQL> Select length(vch) from tmp6  
  
LENGTH (VCH)  
-----  
2
```

可见, 定长字符串在存储时长度是固定的, 而变长字符串的存储长度根据实际插入的数据长度而定。

4.2 如何选择数据类型

Oracle 提供了大量的数据类型, 为了优化存储, 提高数据库性能, 在任何情况下均应使用最精确的类型。即在所有可以表示该列值的类型中, 该类型使用的存储最少。

1. 整数和小数

数值数据类型只有 NUMBER, 但是 NUMBER 功能不小, 它可以存储正数、负数、零、定点数和精度为 30 位的浮点数。格式为 `number(m,n)`, 其中 `m` 为精度, 表示数字的总位数, 它在 1~38; `n` 为范围, 表示小数点右边的数字的位数, 它在 -84~127。

如果不需要小数部分, 则使用整数来保存数据, 可以定义为 `number(m,0)` 或者 `number(m)`; 如果需要表示小数部分, 则使用 `number(m,n)`。

2. 日期与时间类型

如果只需要记录日期, 则可以使用 DATE 类型。如果需要记录日期和时间, 可以使用 TIMESTAMP 类型。特别是需要显示上午、下午或者时区时, 必须使用 TIMESTAMP 类型。

3. CHAR 与 VARCHAR2

CHAR 和 VARCHAR2 的区别:

CHAR 是固定长度字符, VARCHAR2 是可变长度字符; CHAR 会自动补齐插入数据的尾部空格, VARCHAR2 不会补齐尾部空格。

CHAR 是固定长度, 所以它的处理速度比 VARCHAR2 的速度要快, 但是它的缺点就是浪费存储空间。所以对存储不大, 但在速度上有要求的可以使用 CHAR 类型, 反之可以使用 VARCHAR2 类型来实现。

VARCHAR2 虽然比 CHAR 节省空间, 但是如果一个 VARCHAR2 列经常被修改, 而且每次被修改的数据的长度不同, 会引起“行迁移”(Row Migration)现象, 从而造成多余的 I/O, 这是数据库设计和调整中要尽力避免的, 在这种情况下用 CHAR 代替 VARCHAR2 会更好一些。当然还有一种情况就是像身份证号这种长度几乎不变的字段可以考虑使用 CHAR, 以获得更高的效率。

4.3 常见运算符介绍

运算符连接表达式中各个操作数，其作用是用来指明对操作数所进行的运算。常见的运算有数学计算、比较运算、位运算以及逻辑运算。运用运算符可以更加灵活地使用表中的数据，常见的运算符类型有：算术运算符、比较运算符、逻辑运算符、位运算符。本节将介绍各种运算符的特点和使用方法。

4.3.1 运算符概述

运算符是告诉 Oracle 执行特定算术或逻辑操作的符号。Oracle 的内部运算符很丰富，主要有四大类，分别是：算术运算符、比较运算符、逻辑运算符、位操作运算符。

1. 算术运算符

算术运算符用于各类数值运算，包括加（+）、减（-）、乘（*）、除（/）。

2. 比较运算符

比较运算符用于比较运算，包括大于（>）、小于（<）、等于（=）、大于等于（>=）、小于等于（<=）、不等于（!=），以及 IN、BETWEEN AND、IS NULL、LIKE 等。

3. 逻辑运算符

逻辑运算符的求值所得结果均为 1（TRUE）、0（FALSE），这类运算符有逻辑非（NOT 或者!）、逻辑与（AND 或者&&）、逻辑或（OR 或者||）、逻辑异或（XOR）。

4. 位操作运算符

参与运算的操作数，按二进制位进行运算。这类运算符包括位与（&）、位或（|）、位非（~）、位异或（^）、左移（<<）、右移（>>）6 种。

接下来将对 Oracle 中各种运算符的使用进行详细的介绍。

4.3.2 算术运算符

算术运算符是 SQL 中最基本的运算符，Oracle 中的算术运算符如表 4-3 所示。

表 4-3 Oracle 中的算术运算符

运算符	作用
+	加法运算
-	减法运算
*	乘法运算
/	除法运算，返回商

下面分别讨论不同算术运算符的使用方法。

【例 4.9】创建表 tmp7，定义数据类型为 NUMBER 的字段 num，插入值 64，对 num 值进行算术运算。

首先创建表 tmp7，输入语句如下：

```
CREATE TABLE tmp7 ( num NUMBER);
```

向字段 num 插入数据 64：

```
INSERT INTO tmp7 value (64);
```

接下来，对 num 值进行加法和减法运算：

```
SQL> SELECT num, num+10, num-3+5, num+5-3, num+36.5 FROM tmp7;
```

NUM	NUM+10	NUM-3+5	NUM+5-3	NUM+36.5
64	74	66	66	100.5

由计算结果可以看到，可以对 num 字段的值进行加法和减法运算，而且由于“+”和“-”的优先级相同，因此先加后减或者先减后加之后的结果是相同的。

【例 4.10】对 tmp7 表中的 num 进行乘法、除法运算。

```
SQL> SELECT num, num *2, num /2, num/3, num%3 FROM tmp7;
```

NUM	NUM*2	NUM/2	NUM/3
64	128	32	21.33333333

由计算结果可以看到，对 num 进行除法运算的时候，由于 64 无法被 3 整除，因此 Oracle 对 num/3 求商的结果保存到了小数点后面 8 位，结果为 21.33333333。

在数学运算中，除数为 0 的除法是没有意义的，因此除法运算中的除数不能为 0，如果被 0 除，则返回错误提示信息。

【例 4.11】用 0 除 num。

```
SQL> SELECT num / 0 FROM tmp7;
```

错误报告：

SQL 错误：ORA-01476：除数为 0

4.3.3 比较运算符

比较运算符经常在 SELECT 的查询条件子句中使用，用来查询满足指定条件的记录。Oracle 中的比较运算符如表 4-4 所示。

表 4-4 Oracle 中的比较运算符

运算符	作用
=	等于
<=>	安全的等于
<> (!=)	不等于
<=	小于等于
>=	大于等于
>	大于
IS NULL	判断一个值是否为 NULL
IS NOT NULL	判断一个值是否不为 NULL
BETWEEN AND	判断一个值是否落在两个值之间
IN	判断一个值是 IN 列表中的任意一个值
NOT IN	判断一个值不是 IN 列表中的任意一个值
LIKE	通配符匹配

下面分别讨论不同比较运算符的含义。

1. 等于运算符 =

等号“=”用来判断数字、字符串和表达式是否相等。

2. 不等于运算符!=

“!=”用于数字、字符串、表达式不相等的判断。

3. 小于或等于运算符 <=

“<=”用来判断左边的操作数是否小于或者等于右边的操作数。

4. 小于运算符 <

“<”运算符用来判断左边的操作数是否小于右边的操作数。

5. 大于或等于运算符 >=

“>=”运算符用来判断左边的操作数是否大于或者等于右边的操作数。

6. 大于运算符 >

“>”运算符用来判断左边的操作数是否大于右边的操作数。

7. BETWEEN...AND 运算符

BETWEEN...AND 运算符用于测试是否在指定的范围内。通常和 WHERE 子句一起使用，BETWEEN...AND 条件返回一个介于指定上限和下限之间的范围值。

例如下面的例子，选出出生在 1980 到 1990 之间的学生姓名：

```
SELECT name FROM student
WHERE birth BETWEEN '1980' AND '1990';
```

上述语句包含上限值和下限值，和下面的语句效果一样。

```
SELECT name FROM student
WHERE birth>= '1980' AND birth<= '1990';
```

8. IN 运算符

IN 运算符用来判断操作数是否为 IN 列表中的一个值。同样，NOT IN 运算符用来判断操作数是否不是 IN 列表中的一个值。

例如，选出年龄是 25 和 26 的学生。

```
SELECT name FROM student
WHERE age IN (25, 26);
```

9. LIKE

在一个班级中，也许老师只知道某个学生的姓氏，并不知道全部名字。此时就可以使用 LIKE 进行查询。LIKE 运算符用来匹配字符串。

LIKE 运算符在进行匹配时，可以使用下面两种通配符：

- (1) “%”，用来代表由零个或者多个字符组成的任意顺序的字符串。
- (2) “_”，只能匹配一个字符。

例如，选出姓张的所有学生。

```
SELECT name FROM student
WHERE name LIKE '张%';
```

4.3.4 逻辑运算符

Oracle 中的逻辑运算符如表 4-5 所示。

表 4-5 Oracle 中的逻辑运算符

运算符	作用
NOT	逻辑非
AND	逻辑与
OR	逻辑或

这 3 个运算符的作用如下。

(1) NOT 运算符：又称取反运算符，NOT 通常是单目运算符，即 NOT 右侧才能包含表达式，是对结果取反，如果表达式结果为 True，那么 NOT 的结果就为 False；否则，如果表达式的结果为 False，那么 NOT 的结果就为 True。

NOT 运算符常常和 IN、LIKE、BETWEEN...AND 和 NULL 等关键字一起使用。

例如，选择学生年龄不是 25 或者 26 的学生姓名。

```
SELECT name FROM student
WHERE age NOT IN (25,26);
```

(2) AND 运算符：对于 AND 运算符来说，要求两边的表达式结果都为 True，因此通常称为全运算符，如果任何一方的返回结果为 NULL 或 False，那么逻辑运算的结果就为 False，也就是说记录不匹配 WHERE 子句的要求。

例如，选择学生年龄是 25 而且是姓张的学生姓名。

```
SELECT name FROM student
WHERE age=25 AND name LIKE '张%';
```

(3) OR 运算符：OR 运算符又称或运算符，也就是说只要左右两侧的布尔表达式任何一方为 True，结果就为 True。

例如，选择学生年龄是 25 或者姓张的学生姓名。

```
SELECT name FROM student
WHERE age=25 OR name LIKE '张%';
```

这样，无论是年龄为 25 的学生还是姓张的学生，都会被选择出来。

4.3.5 运算符的优先级

运算符的优先级决定了不同的运算符在表达式中计算的先后顺序，表 4-6 列出了 Oracle 中的各类运算符及其优先级。

表 4-6 运算符优先级

优先级	运算符
最低	= (赋值运算), :=
	OR
	AND
	NOT
	= (比较运算), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
	&
	<<, >>
	-, +
	*, /
	- (负号)
最高	!

可以看到,不同运算符的优先级是不同的。一般情况下,级别高的运算符先进行计算,如果级别相同,Oracle 按表达式的顺序从左到右依次计算。当然,在无法确定优先级的情况下,可以使用圆括号()来改变优先级,并且这样会使计算过程更加清晰。

4.4 疑难解惑

疑问 1: Oracle 中如何按照指定格式插入日期和时间?

在插入日期和时间之前,用户需要了解 Oracle 系统默认的时间格式,SQL 语句如下:

```
SQL> select sysdate from dual;
SYSDATE
-----
30-1月-15
```

此时,如果想插入类似“2015-10-13 12:15:30”格式的日期和时间,可以输入以下 SQL 语句:

```
SQL> alter session set nls_date_format='yyyy-mm-dd HH24:mi:ss ';
```

疑问 2: Oracle 中的图片、声音和视频等类型的文件,使用什么格式的数据?

解答: Oracle 中的 blob、clob 和 nclob 三种大型对象,用来保存较大的图形文件或带格式的文本文件,如 Microsoft Word 文档,以及音频、视频等非文本文件,最大长度是 4GB。

4.5 经典习题

- (1) Oracle 中的整数和小数如何表示，不同表示方法之间有什么区别？
- (2) CHAR 和 VARCHAR2 使用时最大的区别是什么？
- (3) 日期用什么类型的数据表示？
- (4) 在 Oracle 中执行如下算术运算： $(9-7)*4$ ， $8+15/3$ ， $17\text{DIV}2$ 。
- (5) 理解 Oracle 中的比较运算： $36>27$ ， $15\geq8$ ， $40<50$ ， $15\leq15$ ， $\text{NULL}\leq\text{NULL}$ ， $\text{NULL}\leq1$ ， $5\leq5$ 的含义。
- (6) 理解 Oracle 中的逻辑运算： $a \text{ AND } b$ ， $a \text{ OR } b$ 和 $\text{NOT } a$ 的含义。

第 5 章

◀ Oracle 函数 ▶

Oracle 提供了众多功能强大、方便易用的函数。使用这些函数，可以极大地提高用户对数据库的管理效率。Oracle 中的函数包括：数学函数、字符串函数、日期和时间函数、条件判断函数、系统信息函数等。本章将介绍 Oracle 中的这些函数的功能和用法。

- 了解什么是 Oracle 函数
- 掌握各种数学函数的用法
- 掌握各种字符串函数的用法
- 掌握时间和日期函数的用法
- 掌握条件函数的用法
- 掌握系统信息函数的用法
- 熟练掌握综合案例中函数的操作方法和技巧

5.1 Oracle 函数简介

函数表示对输入参数值返回一个具有特定关系的值，Oracle 提供了大量丰富的函数，在进行数据库管理以及数据的查询和操作时将会经常用到各种函数。通过对数据的处理，数据库功能可以变得更加强大，更加灵活地满足不同用户的需求。本章将分类介绍不同函数的使用方法。

5.2 数学函数

数学函数主要用来处理数值数据，主要的数学函数有：绝对值函数、三角函数（包括正弦函数、余弦函数、正切函数、余切函数等）、对数函数、随机数函数等。在有错误产生时，数学函数将会返回空值 NULL。本节将介绍各种数学函数的功能和用法。

5.2.1 绝对值函数 ABS(x)

ABS(X)返回 X 的绝对值。

【例 5.1】求 2, -3.3 和 -33 的绝对值, 输入语句如下:

```
SQL> SELECT ABS(2), ABS(-3.3), ABS(-33) FROM dual;
ABS(2)  ABS(-3.3)  ABS(-33)
-----
2         3.3       33
```

正数的绝对值为其本身, 2 的绝对值为 2; 负数的绝对值为其相反数, -3.3 的绝对值为 3.3, -33 的绝对值为 33。



dual 表是一个虚拟表, 用来构成 SELECT 的语法规则, Oracle 保证 dual 里面永远只有一条记录。

5.2.2 平方根函数 SQRT(x)和求余函数 MOD(x,y)

SQRT(x)返回非负数 x 的二次平方根。

【例 5.2】求 9, 40 和 64 的二次平方根, 输入语句如下:

```
SQL> SELECT SQRT(9), SQRT(40), SQRT(64) FROM dual;
SQRT(9)  SQRT(40)  SQRT(64)
-----
3         6.3245532  8
```

3 的平方等于 9, 因此 9 的二次平方根为 3; 40 的平方根为 6.3245532; 64 的平方根为 8。

MOD(x,y)返回 x 被 y 除后的余数, MOD() 对于带有小数部分的数值也起作用, 它返回除法运算后的精确余数。

【例 5.3】对 MOD(31,8), MOD(234, 10), MOD(45.5,6)进行求余运算, 输入语句如下:

```
SQL> SELECT MOD(31,8),MOD(234, 10),MOD(45.5,6) FROM dual;
MOD(31,8) MOD(234,10) MOD(45.5,6)
-----
7         4         3.5
```

5.2.3 获取整数的函数 CEIL(x)和 FLOOR(x)

CEIL(x)返回不小于 x 的最小整数值。

【例 5.4】使用 CEIL 函数返回最小整数, 输入语句如下:

```
SQL> SELECT CEIL(-3.35), CEIL (3.35) FROM dual;
CEIL(-3.35) CEIL(3.35)
-----
```



```
-----
-3              4
```

-3.35 为负数，不小于-3.35 的最小整数为-3，因此返回值为-3；不小于 3.35 的最小整数为 4，因此返回值为 4。

FLOOR(x)返回不大于 x 的最大整数值，返回值转化为一个 BIGINT。

【例 5.5】使用 FLOOR 函数返回最大整数，输入语句如下：

```
SQL> SELECT FLOOR(-3.35), FLOOR(3.35) FROM dual;
FLOOR(-3.35)  FLOOR(3.35)
-----
-4              3
```

-3.35 为负数，不大于-3.35 的最大整数为-4，因此返回值为-4；不大于 3.35 的最大整数为 3，因此返回值为 3。

5.2.4 获取随机数的函数 DBMS_RANDOM.RANDOM 和 DBMS_RANDOM.VALUE (x,y)

DBMS_RANDOM.RANDOM 返回一个随机值。

【例 5.6】使用 DBMS_RANDOM.RANDOM 产生随机数，输入语句如下：

```
SQL>SELECT DBMS_RANDOM.RANDOM , DBMS_RANDOM.RANDOM FROM dual;
RANDOM      RANDOM
-----
-513927314  652092381
```

可以看到，不带参数的 DBMS_RANDOM.RANDOM 每次产生的随机数值是不同的。

【例 5.7】使用 DBMS_RANDOM.VALUE(x,y)函数产生 1~20 之间的随机数，输入语句如下：

```
SQL> SELECT DBMS_RANDOM.VALUE(1,20),DBMS_RANDOM.VALUE(1,20) FROM dual;
DBMS_RANDOM.VALUE(1,20)  DBMS_RANDOM.VALUE(1,20)
-----
1.871131765              4.727097533
```

从结果可以看到，DBMS_RANDOM.VALUE (1,20)产生了 1~20 之间的随机数。

5.2.5 函数 ROUND(x)、ROUND(x,y)和 TRUNC(x,y)

ROUND(x)返回最接近于参数 x 的整数，对 x 值进行四舍五入。

【例 5.8】使用 ROUND(x)函数对操作数进行四舍五入操作，输入语句如下：

```
SQL> SELECT ROUND(-1.14),ROUND(-1.67), ROUND(1.14),ROUND(1.66) FROM dual;
```




```
ROUND(-1.14) ROUND(-1.67) ROUND(1.14) ROUND(1.66)
```

```
-----
```

```
-1          -2          1          2
```

可以看到，四舍五入处理之后，只保留了各个值的整数部分。

ROUND(x,y)返回最接近于参数 x 的数，其值保留到小数点后面 y 位，若 y 为负值，则将保留 x 值到小数点左边 y 位。

【例 5.9】使用 ROUND(x,y)函数对操作数进行四舍五入操作，结果保留小数点后面指定 y 位，输入语句如下：

```
SQL> SELECT ROUND(1.38, 1), ROUND(1.38, 0), ROUND(232.38, -1), ROUND(232.38, -2)
```

```
FROM dual;
```

```
ROUND(1.38,1) ROUND(1.38,0) ROUND(232.38,-1) ROUND(232.38,-2)
```

```
-----
```

```
1.4          1          230          200
```

ROUND(1.38, 1)保留小数点后面 1 位，四舍五入的结果为 1.4；ROUND(1.38, 0) 保留小数点后面 0 位，即返回四舍五入后的整数值；ROUND(232.38, -1)和 ROUND(232.38, -2)分别保留小数点左边 1 位和 2 位。

提示

y 值为负数时，保留的小数点左边的相应位数直接保存为 0，不进行四舍五入。

TRUNC(x,y)返回被舍去至小数点后 y 位的数字 x。若 y 的值为 0，则结果不带有小数点或不带有小数部分。若 y 设为负数，则截去（归零）x 小数点左起第 y 位开始后面所有低位的值。

【例 5.10】使用 TRUNC(x,y)函数对操作数进行四舍五入操作，结果保留小数点后面指定 y 位，输入语句如下：

```
SQL> SELECT TRUNC(1.31, 1), TRUNC(1.99, 1), TRUNC(1.99, 0), TRUNC(19.99, -1) FROM
```

```
dual;
```

```
TRUNC(1.31,1) TRUNC(1.99,1) TRUNC(1.99,0) TRUNC(19.99,-1)
```

```
-----
```

```
1.3          1.9          1          10
```

TRUNC(1.31, 1)和 TRUNC(1.99, 1)都保留小数点后 1 位数字，返回值分别为 1.3 和 1.9；TRUNC(1.99, 0)返回整数部分值 1；TRUNC(19.99, -1)截去小数点左边第 1 位后面的值，并将整数部分的 1 位数字置 0，结果为 10。

提示

ROUND(x,y)函数在截取值的时候会四舍五入，而 TRUNC(x,y)直接截取值，并不进行四舍五入。



5.2.6 符号函数 SIGN(x)

SIGN(x)返回参数的符号，x 的值为负、零或正时返回结果依次为-1、0 或 1。

SIGN(n)函数返回参数 n 的符号。正数返回 1，0 返回 0，负数返回-1。

【例 5.11】返回 3，-10 和 0 的符号，输入语句如下：

```
SQL> SELECT SIGN (3), SIGN (-10), SIGN (0) FROM dual;
SIGN(3)  SIGN(-10)  SIGN(0)
-----  -
1         -1         0
```

SIGN(3)返回 1；SIGN(-10)返回-1；SIGN(0)返回 0。

5.2.7 幂运算函数 POWER(x,y)和 EXP(x)

POWER(x,y)函数返回 x 的 y 次乘方的结果值。

【例 5.12】使用 POWER(x,y)函数进行乘方运算，输入语句如下：

```
SQL> SELECT POWER(2,2), POWER(2,-2) FROM dual;
POWER(2,2)  POWER(2,-2)
-----
4           0.25
```

可以看到，POWER(2,2)返回 2 的 2 次方，结果为 4；POWER(2,-2)返回 2 的-2 次方，结果为 4 的倒数，即 0.25。

EXP(x)返回 e 的 x 乘方后的值。

【例 5.13】使用 EXP(x)函数计算 e 的乘方，输入语句如下：

```
SQL> SELECT EXP (3),EXP (-3),EXP (0) FROM dual;
EXP (3)      EXP (-3)      EXP (0)
-----
20.08553692  0.04978706837      1
```

EXP(3)返回以 e 为底的 3 次方，结果为 20.08553692；EXP(-3)返回以 e 为底的-3 次方，结果为 0.04978706837；EXP(0)返回以 e 为底的 0 次方，结果为 1。

5.2.8 对数运算函数 LOG(x,y)和 LN(x)

LOG(x, y)返回以 x 为底 y 的对数。

【例 5.14】使用 LOG(x,y)函数计算自然对数，输入语句如下：

```
SQL>SELECT LOG(10,100), LOG(7,49) FROM dual;
LOG(10,100)  LOG(7,49)
-----
```




2

2

10 的 2 次乘方等于 100，因此 LOG(10,100)返回结果为 2；同样，LOG(7,49)返回结果为 2。
LN(x)返回 x 的自然对数，即 x 相对于基数 e 的对数，参数 x 要求大于 0。

【例 5.15】使用 LN(x)函数计算以 e 为基数的对数，输入语句如下：

```
SQL> SELECT LN(2), LN(100) FROM dual;
LN(2)          LN(100)
-----
0.6931471806   4.605170186
```

5.2.9 正弦函数 SIN(x)和反正弦函数 ASIN(x)

SIN(x)返回 x 的正弦，其中 x 为弧度值。

【例 5.16】使用 SIN 函数计算正弦值，输入语句如下：

```
SQL> SELECT SIN(1), SIN(3) FROM dual;
SIN(1)          SIN(3)
-----
0.8414709848    0.1411200081
```

ASIN(x)返回 x 的反正弦，即正弦为 x 的值。x 的取值在-1~1 的范围之内。

【例 5.17】使用 ASIN 函数计算反正弦值，输入语句如下：

```
SQL> SELECT ASIN(0.8414709848), ASIN(0.1411200081) FROM dual;
ASIN(0.8414709848)  ASIN(0.1411200081)
-----
1                   3
```

由结果可以看到，函数 ASIN 和 SIN 互为反函数。

5.2.10 余弦函数 COS(x)和反余弦函数 ACOS(x)

COS(x)返回 x 的余弦，其中 x 为弧度值。

【例 5.18】使用 COS 函数计算余弦值，输入语句如下：

```
SQL> SELECT COS(0), COS(1) FROM dual;
COS(0)          COS(1)
-----
1               0.5403023059
```

由结果可以看到，COS(0)值为 1；COS(1)值为 0.5403023059。

ACOS(x)返回 x 的反余弦，即余弦是 x 的值。x 的取值在-1~1 的范围之内。



【例 5.19】使用 ACOS 函数计算反余弦值，输入语句如下：

```
SQL> SELECT ACOS(1),ACOS(0.5403023059) FROM dual;
ACOS(1)  ACOS(0.5403023059)
-----
0          1
```

由结果可以看到，函数 ACOS 和 COS 互为反函数。

5.2.11 正切函数 TAN(x)和反正切函数 ATAN(x)

TAN(x)返回 x 的正切，其中 x 为给定的弧度值。

【例 5.20】使用 TAN 函数计算正切值，输入语句如下：

```
SQL> SELECT TAN(0.3), TAN(0.7853981634) FROM dual;
TAN(0.3)          TAN(0.7853981634)
-----
0.3093362496          1
```

ATAN(x)返回 x 的反正切，即正切为 x 的值。

【例 5.21】使用 ATAN 函数计算反正切值，输入语句如下：

```
SQL> SELECT ATAN(0.3093362496), ATAN(1) FROM dual;
ATAN(0.3093362496)  ATAN(1)
-----
0.3          0.7853981634
```

由结果可以看到，函数 ATAN 和 TAN 互为反函数。

5.3 字符串函数

字符串函数主要用来处理数据库中的字符串数据。本节将介绍各种字符串函数的功能和用法。

5.3.1 计算字符串长度的函数

LENGTH(str)返回值为字符串的字节长度。

【例 5.22】使用 LENGTH 函数计算字符串长度，输入语句如下：

```
SQL> SELECT LENGTH('date'), LENGTH('egg') FROM dual ;
LENGTH('DATE')  LENGTH('EGG')
```




4

3

5.3.2 合并字符串函数 CONCAT(s1,s2)

CONCAT(s1,s2)返回结果为连接参数产生的字符串。

【例 5.23】使用 CONCAT 函数连接字符串，输入语句如下：

```
SQL> SELECT CONCAT('学习', 'Oracle 12c') FROM dual;
CONCAT('学习','ORACLE 12C')
-----
学习 Oracle 12c
```

CONCAT('学习', 'Oracle 12c') 返回两个字符串连接后的字符串。

5.3.3 字符串搜索函数 INSTR(s,x)

INSTR(s,x)返回 x 字符在字符串 s 的位置。

【例 5.24】使用 INSTR 函数返回字符在字符串的位置，输入语句如下：

```
SQL> SELECT INSTR('hello Oracle', 'c') FROM dual;
INSTR('HELLOORACLE','C')
-----
10
```

字符 c 位于字符串'hello Oracle'的第 10 个位置，结果输出为 10。

5.3.4 字母大小写转换函数

LOWER(str)可以将字符串 str 中的字母字符全部转换成小写字母。

【例 5.25】使用 LOWER 函数将字符串中所有字母字符转换为小写，输入语句如下：

```
SQL> SELECT LOWER('BEAUTIFUL') FROM dual ;
LOWER('BEAUTIFUL')
-----
beautiful
```

由结果可以看到，原来所有字母为大写的，全部转换为小写，如“BEAUTIFUL”，转换之后为“beautiful”。

UPPER(str)可以将字符串 str 中的字母字符全部转换成大写字母。

【例 5.26】使用 UPPER 函数将字符串中所有字母字符转换为大写，输入语句如下：

```
SQL> SELECT UPPER('black') FROM dual;
UPPER('BLACK')
```



```
-----  
BLACK
```

由结果可以看到，原来所有字母字符为小写的，全部转换为大写，如“black”，转换之后为“BLACK”。

INITCAP(str) 将输入的字符串单词的首字母转换成大写。如果不是两个字母连在一起，则认为是新的单词，类似 a_b a,b a b 这些情况，a 和 b 都会转换成大写。

【例 5.27】使用 INITCAP 函数将字符串中首字母转换成大写，输入语句如下：

```
SQL> SELECT INITCAP ('hello beautiful word ') FROM dual;  
INITCAP('HELLOBEAUTIFULWORD')  
-----  
Hello Beautiful Word
```

由结果可以看到，原来每个单词的首字母，全部转换为大写，如“hello”，转换之后为“Hello”。

5.3.5 获取指定长度的字符串的函数 SUBSTR(s,m,n)

SUBSTR(s,m,n)函数用于获取指定的字符串。其中参数 s 代表字符串，m 代表截取的位置，n 代表截取长度。

【例 5.28】使用 SUBSTR 函数返回字符串中指定的字符，输入语句如下：

```
SQL>SELECT SUBSTR ('abcdf 好 efgf', 6,2), SUBSTR ('abcdf 好 efgf',-6,2) FROM  
dual;  
SUBSTR('ABCDF 好 EFGF',6,2) SUBSTR('ABCDF 好 EFGF',-6,2)  
-----  
好 e f 好
```

当 m 值为正数时，从左边开始取指定位置的字符；当 m 值为负数时，从右边开始取指定位置的字符。

5.3.6 替换字符串的函数 REPLACE(s1,s2,s3)

REPLACE (s1,s2,s3)是一个替换字符串的函数。其中参数 s1 表示搜索的目标字符串；s2 表示在目标字符串中要搜索的字符串；s3 是可选参数，用它替换被搜索到的字符串，如果不用该参数，表示从 s1 字符串中删除搜索到的字符串。

【例 5.29】使用 REPLACE 函数对字符串进行替换操作，输入语句如下：

```
SQL>SELECT REPLACE ('this is a dog', 'dog','cat') , ('this is a dog', 'dog')  
FROM dual;  
REPLACE('THISISADOG','DOG','CAT') REPLACE('THISISADOG','DOG')  
-----
```




```
this is a cat
```

```
this is a
```

从结果可以看出，第一个替换的情况是：字符串“dog”被替换成“cat”；第二个替换的情况是：字符串“dog”被删除。

5.3.7 删除字符串首尾指定字符的函数 LTRIM(s,n)和 RTRIM(s,n)

LTRIM(s,n)函数将删除指定的左侧字符。其中 s 是目标字符串，n 是需要查找的字符。如果 n 不指定，则表示删除左侧的空格。

【例 5.30】使用 LTRIM 函数对字符串进行删除操作，输入语句如下：

```
SQL>SELECT LTRIM ('this is a dog', 'this') , LTRIM ('  this is a dog') FROM
dual;
LTRIM('THISISADOG','THIS') LTRIM('THISISADOG')
-----
is a dog                this is a dog
```

从结果可以看出，第一个删除的情况是：字符串的“this”字符被删除；第二个删除的情况是：字符串左侧的空格被删除。

RTRIM(s,n) 函数将删除指定的右侧字符。其中 s 是目标字符串，n 是需要查找的字符。如果 n 不指定，则表示删除右侧的空格。

【例 5.31】使用 RTRIM 函数对字符串进行删除操作，输入语句如下：

```
SQL>SELECT RTRIM ('this is a dog', 'dog') , RTRIM (' this is a dog  ') FROM
dual;
RTRIM('THISISADOG','DOG') RTRIM('THISISADOG')
-----
this is a                this is a dog
```

从结果可以看出，第一个删除的情况是：字符串的“dog”字符被删除；第二个删除的情况是：字符串右侧的空格被删除。

5.3.8 删除指定字符串的函数 TRIM()

TRIM 函数将删除指定的前缀或者后缀的字符，默认删除空格。具体的语法格式如下：

```
TRIM ([LEADING/TRAILING/BOTH] [trim_character FROM] trim_source)
```

其中 LEADING 指删除 trim_source 的前缀字符；TRAILING 指删除 trim_source 的后缀字符；BOTH 指删除 trim_source 的前缀和后缀字符；trim_character 指删除的指定字符，默认删除空格；trim_source 指被操作的源字符串。

【例 5.32】使用 TRIM(s1 FROM s)函数删除字符串中两端指定的字符，输入语句如下：

```
SQL> SELECT TRIM( BOTH 'x' FROM 'xyxbxykyx'), TRIM('  xyxyxy  ') FROM dual;
TRIM(BOTH'X'FROM'YXBXKYKX') TRIM('XYXYXY')
```



```
-----  
yxbxyky
```

```
xyxyxy
```

删除字符串 “xyxbxykyx” 两端的重复字符 “x”，而中间的 “x” 并不删除，结果为 “yxbxyky”。

5.3.9 字符集名称和 ID 互换函数

NLS_CHARSET_ID(string)函数可以得到字符集名称对应的 ID。参数 string 表示字符集的名称。

【例 5.33】使用 NLS_CHARSET_ID 函数获取字符集名称，输入语句如下：

```
SQL>SELECT NLS_CHARSET_ID('US7ASCII') FROM dual;  
NLS_CHARSET_ID('US7ASCII')  
-----  
1
```

NLS_CHARSET_NAME(number)函数可以得到字符集 ID 对应的名称。参数 number 表示字符集的 ID。

【例 5.34】使用 NLS_CHARSET_NAME 函数获取字符集名称，输入语句如下：

```
SQL>SELECT NLS_CHARSET_NAME(1) FROM dual;  
NLS_CHARSET_NAME(1)  
-----  
US7ASCII
```

5.4 日期和时间函数

日期和时间函数主要用来处理日期和时间值，一般的日期函数除了使用 DATE 类型的参数外，也可以使用 TIMESTAMP 类型的参数，但会忽略这些值的时间部分。本节将介绍各种日期和时间函数的功能和用法。

5.4.1 获取当前日期和时间的函数

SYSDATE()函数获取当前系统日期。

【例 5.35】使用日期函数获取系统当前日期，输入语句如下：

```
SQL> SELECT SYSDATE FROM dual;  
SYSDATE  
-----
```




```
01-2月 -15
```

【例 5.36】使用日期函数获取指定格式的系统当前日期，输入语句如下：

```
SQL> SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD HH24:MI:SS') FROM dual;
TO_CHAR(SYSDATE, 'YYYY-MM-DDHH24:MI:SS')
-----
2015-02-01 12:25:28
```

SYSTIMESTAMP()函数获取当前系统时间，该时间包含时区信息，精确到微秒。返回类型为带时区信息的 TIMESTAMP 类型。

【例 5.37】使用日期函数获取系统当前时间，输入语句如下：

```
SQL> SELECT SYSTIMESTAMP FROM dual;
SYSTIMESTAMP
-----
01-2月 -15 12.29.55.714000000 下午 +08:00
```

5.4.2 获取时区的函数

DBTIMEZONE 函数返回数据库所在的时区。

【例 5.38】使用 DBTIMEZONE 函数获取数据库所在的时区，输入语句如下：

```
SQL> SELECT DBTIMEZONE FROM dual;
DBTIMEZONE
-----
+00:00
```

SESSIONTIMEZONE 函数返回当前会话所在的时区。

【例 5.39】使用 SESSIONTIMEZONE 函数获取当前会话所在的时区，输入语句如下：

```
SQL> SELECT SESSIONTIMEZONE FROM dual;
SESSIONTIMEZONE
-----
Asia/Shanghai
```

5.4.3 获取指定月份最后一天的函数

LAST_DAY(date)函数返回参数指定日期对应月份的最后一天。

【例 5.40】使用 LAST_DAY 函数返回指定月份最后一天，输入语句如下：

```
SQL> SELECT LAST_DAY(SYSDATE) FROM dual;
LAST_DAY(SYSDATE)
```



```
-----
28-2月 -15
```

LAST_DAY(SYSDATE)返回 2 月份的最后一天 2015 年 2 月 28 日。

5.4.4 获取指定日期后一周的日期的函数

NEXT_DAY(date,char)函数获取指定日期向后一周的对应日期，char 表示是星期几，全称和缩写都允许，但必须是有效值。

【例 5.41】使用 NEXT_DAY 函数返回指定日期后一周的日期。输入语句如下：

```
SQL> SELECT NEXT_DAY (SYSDATE, '星期日') FROM dual;
SYSDATE      NEXT_DAY(SYSDATE, '星期日')
-----
01-2月-15     08-2月-15
```

NEXT_DAY (SYSDATE, '星期日')返回当前日期后第一个周日的日期。

5.4.5 获取指定日期特定部分的函数

EXTRACT(datetime)函数可以从指定的时间中提取特定部分。例如，提取年份、月份或者时等。

【例 5.42】使用 EXTRACT 函数获取年份等特定部分。输入语句如下：

```
SQL> SELECT EXTRACT (YEAR FROM SYSDATE), EXTRACT (MINUTE FROM TIMESTAMP
'1985-10-8 12:23:40') FROM dual;

EXTRACT(YEARFROMSYSDATE) EXTRACT(MINUTEFROMTIMESTAMP'1985-10-812:23:40')
-----
2015                      23
```

从结果可以看出，分别返回了年份和分钟。

5.4.6 获取两个日期之间的月份数的函数

MONTHS_BETWEEN(date1,date2)函数返回 date1 和 date2 之间的月份数。

【例 5.43】使用 MONTHS_BETWEEN 函数获取两个日期之间的月份数。输入语句如下：

```
SQL>SELECT
MONTHS_BETWEEN(TO_DATE('1985-10-8','YYYY-MM-DD'),TO_DATE('1985-8-8','YYYY
-MM-DD'),one,
MONTHS_BETWEEN(TO_DATE('1985-05-8','YYYY-MM-DD'),TO_DATE('1985-07-8','YYY
```



```
Y-MM-DD') ) TEO FROM dual;
```

```
ONE          TEO
```

```
-----
```

```
2           -2
```

从结果可以看出，当 `date1>date2` 时，返回数值为一个整数；当 `date1<date2` 时，返回数值为一个负数。

5.5 转换函数

转换函数的主要作用是完成不同数据类型之间的转换。本节将分别介绍各个转换函数的用法。

5.5.1 字符串转 ASCII 类型字符串函数

`ASCIISTR(char)` 函数可以将任意字符串转换为数据库字符集对应的 ASCII 字符串。`char` 为字符类型。

【例 5.44】使用 `ASCIISTR` 函数把字符串转为 ASCII 类型。输入语句如下：

```
SQL> SELECT ASCIISTR('从零开始学') FROM dual;
```

```
ASCIISTR('从零开始学')
```

```
-----
```

```
\4ECE\96F6\5F00\59CB\5B66
```

5.5.2 二进制转十进制函数

`BIN_TO_NUM()` 函数可以实现将二进制转换成对应的十进制。

【例 5.45】使用 `BIN_TO_NUM` 函数把二进制转换为十进制类型。输入语句如下：

```
SQL> SELECT BIN_TO_NUM (1,1,0) FROM dual;
```

```
BIN_TO_NUM(1,1,0)
```

```
-----
```

```
6
```

5.5.3 数据类型转换函数

在 Oracle 中，用户如果想把数字转换为字符或者字符转换为日期，通常使用 `CAST(expr as type_name)` 函数来完成。

【例 5.46】使用 `CAST` 函数在数字与字符串之间进行转换操作。输入语句如下：

```
SQL> SELECT CAST ('4321' AS NUMBER) , CAST (4321 AS VARCHAR2) FROM dual;
CAST ('4321' AS NUMBER)    CAST (4321 AS VARCHAR2)
-----
4321                        4321
```

5.5.4 数值转换为字符串函数

TO_CHAR 函数将一个数值型参数转换成字符型数据。具体语法格式如下：

```
TO_CHAR(n, [fmt[nlsparam]])
```

其中参数 n 代表数值型数据；参数 fmt 代表要转换成字符的格式；nlsparam 参数代表指定 fmt 的特征，包括小数点字符、组分隔符和本地钱币符号。

【例 5.47】使用 TO_CHAR 函数把数值类型转换为字符串。输入语句如下：

```
SQL> SELECT TO_CHAR (10.13245, '99.999'), TO_CHAR (10.13245) FROM dual;
TO_CHAR(10.13245,'99.999') TO_CHAR(10.13245)
-----
10.132                      10.13245
```

由结果可知，如果不指定转换的格式，则数值直接转换为字符串，不做任何格式处理。另外，TO_CHAR 函数还可以将日期类型转换为字符串类型。

【例 5.48】使用 TO_CHAR 函数把日期类型转换为字符串类型。输入语句如下：

```
SQL> SELECT TO_CHAR (SYSDATE, 'YYYY-MM-DD'), TO_CHAR (SYSDATE, 'HH24-MI-SS')
FROM dual;
TO_CHAR(SYSDATE,'YYYY-MM-DD') TO_CHAR(SYSDATE,'HH24-MI-SS')
-----
2015-02-05                      12-06-49
```

5.5.5 字符转日期函数

TO_DATE 函数将一个字符型数据转换成日期型数据。具体语法格式如下：

```
TO_DATE(char[,fmt[,nlsparam]])
```

其中参数 char 代表需要转换的字符串；参数 fmt 代表要转换成字符的格式；nlsparam 参数控制格式化时使用的语言类型。

【例 5.49】使用 TO_DATE 函数把字符串类型转换为日期类型。输入语句如下：

```
SQL> SELECT TO_CHAR(TO_DATE ('1999-10-16', 'YYYY-MM-DD'),'MONTH') FROM dual;
TO_CHAR(TO_DATE('1999-10-16','YYYY-MM-DD'),'MONTH')
-----
10月
```


5.5.6 字符串转数字函数

TO_NUMBER 函数将一个字符型数据转换成数字数据。具体语法格式如下：

```
TO_NUMBER (expr[,fmt[,nlsparam]])
```

其中参数 `expr` 代表需要转换的字符串；参数 `fmt` 代表要转换成数字的格式；`nlsparam` 参数指定 `fmt` 的特征，包括小数点字符、组分隔符和本地钱币符号。

【例 5.50】使用 TO_NUMBER 函数把字符串类型转换为数字类型。输入语句如下：

```
SQL> SELECT TO_NUMBER ('1999.123', '9999.999') FROM dual;
TO_NUMBER('1999.123','9999.999')
-----
1999.123
```

5.6 系统信息函数

本节将介绍常用的系统信息函数，Oracle 中的系统信息函数有：返回登录名函数和返回会话以及上下文信息函数等。

5.6.1 返回登录名函数

USER 函数返回当前会话的登录名。

【例 5.51】使用 USER 函数返回当前会话的登录名称。输入语句如下：

```
SQL> SELECT USER FROM dual;
USER
-----
SYS
```

5.6.2 返回会话以及上下文信息函数

USERENV 函数返回当前会话的信息。具体的语法格式如下：

```
USERENV (parameter)
```

当参数为 `Language` 时，返回会话对应的语言、字符集等；当参数为 `SESSION` 时，返回当前会话的 ID；当参数为 `ISDBA` 时，返回当前用户是否为 DBA。

【例 5.52】使用 USERENV 函数返回当前会话对应的语言和字符集等信息。输入语句如下：

```
SQL> SELECT USERENV('Language') FROM dual;
USERENV('LANGUAGE')
```

SIMPLIFIED CHINESE_CHINA.ZHS16GBK

5.7 综合案例——Oracle 函数的使用

本章为读者介绍了大量的 Oracle 函数，包括数学函数、字符串函数、日期和时间函数、系统函数。读者应该在实践过程中深入了解、掌握这些函数。不同版本的 Oracle 之间的函数可能会有微小的差别，使用时需要查阅对应版本的参考手册，但大部分函数功能在不同版本的 Oracle 之间是一致的。接下来将给出一个使用各种 Oracle 函数的综合案例。

1. 案例目的

使用各种函数操作数据，掌握各种函数的作用和使用方法。

2. 案例操作过程

步骤 01 使用数学函数 DBMS_RANDOM.VALUE ()生成两个 10 以内的随机数。

数学函数 DBMS_RANDOM.VALUE(x,y)产生 1~10 的随机数，输入语句如下：

```
SQL> SELECT DBMS_RANDOM.VALUE(1,10),DBMS_RANDOM.VALUE(1,10) FROM dual;

DBMS_RANDOM.VALUE(1,10) DBMS_RANDOM.VALUE(1,10)
-----
7.871131765                2.727097533
```

由结果可以看到，DBMS_RANDOM.VALUE(1,10)产生了 1~10 的随机数。

步骤 02 使用 SIN(), COS(), TAN()函数计算三角函数值，并将计算结果转换成整数值。

Oracle 中的三角函数可以计算正弦、余弦和正切，执行过程如下：

```
SQL> SELECT SIN(2), COS(1), TAN(0.3), TAN(1.3) FROM dual;

SIN(2)      COS(1)      TAN(0.3)      TAN(1.3)
-----
0.9092974268 0.5403023059 0.3093362496 3.602102448
```

步骤 03 创建表，并使用字符串和日期函数，对字段值进行操作。

(1) 创建表 member，其中包含 5 个字段，分别为 GENERATED BY DEFAULT AS IDENTITY 约束的 m_id 字段，VARCHAR2 类型的 m_FN 字段，VARCHAR2 类型的 m_LN 字段，DATE 类型的 m_birth 字段和 VARCHAR2 类型的 m_info 字段。

(2) 插入一条记录，m_id 值为默认，m_FN 值为“Halen”，m_LN 值为“Park”，m_birth 值为“1970-06-29”，m_info 值为“GoodMan”。

(3) 返回 m_FN 的长度，返回第一条记录中的人的全名，将 m_info 字段值转换成小写字母，将 m_info 的值反向输出。

(4) 计算第一条记录中人的年龄，并计算 m_birth 字段中的值在那一年中的位置，按照“YYYY-MM-DD”格式输出时间值。

(5) 插入一条新的记录，m_FN 值为“Samuel”，m_LN 值为“Green”，m_birth 值为系统当前时间，m_info 值为“GoodWoman”。使用 NLS_CHARSET_ID() 查看最后插入的 ID 值。

操作过程如下：

(1) 创建表 member，输入语句如下：

```
CREATE TABLE member
(
  m_id    NUMBER(11) GENERATED BY DEFAULT AS IDENTITY,
  m_FN    VARCHAR2(100),
  m_LN    VARCHAR2(100),
  m_birth DATE,
  m_info  VARCHAR2(255) NULL
);
```

(2) 插入一条记录，输入语句如下：

```
INSERT INTO member VALUES (1011, 'Halen ', 'Park', '29-6月-1970', 'GoodMan');
```

使用 SELECT 语句查看插入结果：

```
SQL> SELECT * FROM member;
M_ID  M_FN  M_LN  M_BIRTH  M_INFO
-----
1011  Halen  Park   29-6月-70  GoodMan
```

(3) 返回 m_FN 的长度，返回第一条记录中的人的全名，将 m_info 字段值转换成小写字母，将 m_info 的值全部改为大写输出。输入 SQL 语句，执行结果如下：

```
SELECT LENGTH(m_FN), CONCAT(m_FN, m_LN),
       LOWER(m_info), UPPER (m_info) FROM member;
LENGTH(M_FN)  CONCAT (M_FN,M_LN)  LOWER(M_INFO)  UPPER (M_INFO)
-----
5              HalenPark          goodman        GOODMAN
```

(4) 计算第一条记录中人的年龄，并计算 m_birth 字段中的值在那一年中的位置，按照“YYYY-MM-DD”格式输出时间值。

```
SELECT YEAR(CURDATE())-YEAR(m_birth) AS age, YEAR(m_birth) AS days,
TO_CHAR(m_birth, 'YYYY-MM-DD') AS birth FROM member;
```

语句执行结果如下：

```
AGE  DAYS  BIRTH
-----
45    1970    1970-06-29
```

(5) 插入一条新的记录，m_FN 值为“Samuel”，m_LN 值为“Green”，m_birth 值为系统当前时间，m_info 值为“GoodWoman”。使用 HLS_CHARSET_ID() 查看最后插入的 ID 值。

```
INSERT INTO member VALUES (1012, 'Samuel', 'Green', SYSDATE, 'GoodWoman');
```

使用 SELECT 语句查看插入结果：

```
SQL> SELECT * FROM member;
M_ID M_FN  M_LN  M_BIRTH  M_INFO
-----
1011 Halen   Park   29-6月-70  GoodMan
1012 Samuel  Green  5-2月-15  GoodWoman
```

可以看到，表中现在有两条记录，接下来使用 NLS_CHARSET_ID() 函数查看最后插入的 ID 值，输入语句如下：

```
SQL> SELECT COUNT(*) FROM member;
COUNT(*)
-----
2
```

最后插入的为第二条记录，因此返回值为 2。

5.8 疑难解惑

疑问 1：如何从日期时间值中获取年、月、日等部分日期或时间值？

Oracle 中，日期时间值以字符串形式存储在数据表中，因此可以使用字符串函数分别截取日期时间值的不同部分，例如某个名称为 dt 的字段有值“1985-10-08 12:23:40”，如果只需要获得年值，可以输入 YEAR FROM TIMESTAMP '1985-10-8 12:23:40'；如果只需要获得月份值，可以输入 MCNTH FROM TIMESTAMP '1985-10-8 12:23:40'。

疑问 2：如何选择列表中第一个不为空的表达式？

COALESCE(expr) 函数返回列表中第一个不为 NULL 的表达式。如果全部为 NULL，则返

回一个 NULL。例如以下例子：

```
SQL> SELECT COALESCE (1-2,NULL ,9-8,NULL) FROM dual;
COALESCE (NULL,9-8,NULL)
-----
-1
```

5.9 经典习题

1. 使用数学函数进行如下运算：

- (1) 计算 18 除以 5 的商和余数。
- (2) 将弧度值 0.125 转换为角度值。
- (3) 计算 9 的 4 次方值。
- (4) 保留浮点值 3.14159 小数点后面 2 位。

2. 使用字符串函数进行如下运算：

- (1) 分别计算字符串 “Hello World!” 和 “University” 的长度。
- (2) 从字符串 “Nice to meet you!” 中获取子字符串 “meet”。
- (3) 重复输出 3 次字符串 “Cheer!”。
- (4) 将字符串 “voodoo” 逆序输出。
- (5) 4 个字符串 “Oracle”、“not”、“is”、“great”，按顺序排列，从中选择 1、3 和 4 位置处的字符串组成新的字符串。

3. 使用日期和时间函数进行如下运算：

- (1) 计算当前日期是一年的第几周。
- (2) 计算当前日期是一周中的第几个工作日。
- (3) 计算 “1929-02-14” 与当前日期之间相差的年份。
- (4) 按 “97 Oct 4th Saturday” 格式输出当前日期。
- (5) 从当前日期时间值中获取时间值，并将其转换为秒值。

第 6 章

◀ 查询数据 ▶

数据库管理系统的一个最重要的功能就是数据查询，数据查询不应只是简单返回数据库中存储的数据，还应该根据需要对数据进行筛选，以及确定数据以什么样的格式显示。Oracle 提供了功能强大、灵活的语句来实现这些操作，本章将介绍如何使用 SELECT 语句查询数据表中的一列或多列数据、使用集合函数显示查询结果、连接查询、子查询以及使用正则表达式进行查询等。

- 了解基本查询语句
- 掌握表单查询的方法
- 掌握如何使用几何函数查询
- 掌握连接查询的方法
- 掌握如何使用子查询
- 熟悉合并查询结果
- 熟悉如何为表和字段取别名
- 掌握如何使用正则表达式查询
- 掌握综合案例中数据表的查询操作技巧和方法

6.1 基本查询语句

Oracle 从数据表中查询数据的基本语句为 SELECT 语句。SELECT 语句的基本格式是：

```
SELECT
    { * | <字段列表> }
    [
        FROM <表1>,<表2>...
        [WHERE <表达式>]
        [GROUP BY <group by definition>]
        [HAVING <expression> [{<operator> <expression>}...]]
```



```

        [ORDER BY <order by definition>]

        [LIMIT [<offset>,<row count>]

    ]

SELECT [字段1, 字段2, ..., 字段 n]

FROM [表或视图]

WHERE [查询条件];

```

其中，各条子句的含义如下：

- `{*|<字段列表>}` 包含星号通配符选字段列表，表示查询的字段，其中字段列至少包含一个字段名称，如果要查询多个字段，多个字段之间用逗号隔开，最后一个字段后不要加逗号。
- `FROM <表 1>,<表 2>...`，表 1 和表 2 表示查询数据的来源，可以是单个或者多个。
- `WHERE` 子句是可选项，如果选择该项，将限定查询行必须满足的查询条件。
- `GROUP BY <字段>`，该子句告诉 Oracle 如何显示查询出来的数据，并按照指定的字段分组。
- `[ORDER BY <字段>]`，该子句告诉 Oracle 按什么样的顺序显示查询出来的数据，可以进行的排序有：升序（ASC）、降序（DESC）。
- `[LIMIT [<offset>,<row count>]`，该子句告诉 Oracle 每次显示查询出来的数据条数。

`SELECT` 的可选参数比较多，读者可能无法一下完全理解，不要紧，接下来将从最简单的开始，一步一步深入学习之后，读者会对各个参数的作用有清晰的认识。

下面以一个例子说明如何使用 `SELECT` 从单个表中获取数据。

首先定义数据表，输入语句如下：

```

CREATE TABLE fruits
(
    f_id    varchar2(10)    NOT NULL,
    s_id    number(6)       NOT NULL,
    f_name  varchar2(255)   NOT NULL,
    f_price number (8,2)    NOT NULL
);

```

为了演示如何使用 `SELECT` 语句，需要插入如下数据：

```

INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('a1',
101, 'apple', 5.2);

INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('b1', 101, 'blackberry',
10.2);

```



```

INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('bs1',102,'orange',
11.2);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('bs2',105,'melon',8.2);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('t1',102,'banana',
10.3);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('t2',102,'grape', 5.3);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('o2',103,'coconut',
9.2);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('c0',101,'cherry',
3.2);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('a2',103,
'apricot',2.2);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('l2',104,'lemon', 6.4);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('b2',104,'berry', 7.6);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('m1',106,'mango',
15.6);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('m2',105,'xbabay',
2.6);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('t4',107,'xbababa',
3.6);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('m3',105,'xxtt',
11.6);
INSERT INTO fruits (f_id, s_id, f_name, f_price) VALUES ('b5',107,'xxxx', 3.6);

```

使用 SELECT 语句查询 f_id 和 f_name 字段的数据。

```
SQL> SELECT f_id, f_name FROM fruits;
```

```
F_ID      F_NAME
```

```
-----
```

```
a1        apple
```

```
b1        blackberry
```

```
bs1       orange
```

```
bs2       melon
```

```
t1        banana
```

```
t2        grape
```



```

o2      coconut
c0      cherry
a2      apricot
l2      lemon
b2      berry
m1      mango
m2      xbabay
t4      xbababa
m3      xxtt
b5      xxxx

```

该语句的执行过程是, SELECT 语句决定了要查询的列值, 在这里查询 f_id 和 f_name 两个字段的值, FROM 子句指定了数据的来源, 这里指定数据表 fruits, 因此返回结果为 fruits 表中 f_id 和 f_name 两个字段下所有的数据。其显示顺序为添加到表中的顺序。

6.2 单表查询

单表查询是指从一张数据表中查询所需的数据。本节将介绍单表查询中的各种基本的查询方式, 主要有: 查询所有字段、查询指定字段、查询指定记录、查询空值、多条件查询、对查询结果进行排序等。

6.2.1 查询所有字段

1. 在 SELECT 语句中使用星号 "*" 通配符查询所有字段

SELECT 查询记录最简单的形式是从一个表中检索所有记录, 实现的方法是使用星号 (*) 通配符指定查找所有列的名称。语法格式如下:

```
SELECT * FROM 表名;
```

【例 6.1】从 fruits 表中检索所有字段的数据, SQL 语句如下:

```

SQL> SELECT * FROM fruits;

f_id  s_id  f_name  f_price
-----
a1    101   apple   5.20
a2    103   apricot  2.20
b1    101   blackberry 10.20
b2    104   berry    7.60

```

b5	107	xxxx	3.60
bs1	102	orange	11.20
bs2	105	melon	8.20
c0	101	cherry	3.20
l2	104	lemon	6.40
m1	106	mango	15.60
m2	105	xbabay	2.60
m3	105	xxtt	11.60
o2	103	coconut	9.20
t1	102	banana	10.30
t2	102	grape	5.30
t4	107	xbababa	3.60

可以看到，使用星号（*）通配符时，将返回所有列，列按照定义表的时候的顺序显示。

2. 在 SELECT 语句中指定所有字段

下面介绍另外一种查询所有字段值的方法。根据前面 SELECT 语句的格式，SELECT 关键字后面的字段名为将要查找的数据，因此可以将表中所有字段的名称写在 SELECT 子句后面，如果忘记了字段名称，可以使用 DESC 命令查看表的结构。有时候，由于表中的字段可能比较多，不一定能记得所有字段的名称，因此该方法会很不方便，不建议使用。例如查询 fruits 表中的所有数据，SQL 语句也可以书写如下：

```
SELECT f_id, s_id, f_name, f_price FROM fruits;
```

查询结果与【例 6.1】相同。

提示

一般情况下，除非需要使用表中所有的字段数据，最好不要使用通配符“*”。使用通配符虽然可以节省输入查询语句的时间，但是获取不需要的列数据通常会降低查询和所使用的应用程序的效率。通配符的优势是，当不知道所需要的列的名称时，可以通过它获取它们。

6.2.2 查询指定字段

1. 查询单个字段

查询表中的某一个字段，语法格式为：

```
SELECT 列名 FROM 表名;
```

【例 6.2】查询 fruits 表中 f_name 列所有水果名称，SQL 语句如下：


```
SELECT f_name FROM fruits;
```

该语句使用 SELECT 声明从 fruits 表中获取名称为 f_name 字段下的所有水果名称，指定字段的名称紧跟在 SELECT 关键字之后，查询结果如下：

```
SQL> SELECT f_name FROM fruits;
```

```
f_name
apple
apricot
blackberry
berry
xxxx
orange
melon
cherry
lemon
mango
xbabay
xxtt
coconut
banana
grape
xbababa
```

输出结果显示了 fruits 表中 f_name 字段下的所有数据。

2. 查询多个字段

使用 SELECT 声明，可以获取多个字段下的数据，只需要在关键字 SELECT 后面指定要查找的字段名称，不同字段名称之间用逗号(,)分隔开，最后一个字段后面不需要加逗号，语法格式如下：

```
SELECT 字段名1, 字段名2, ..., 字段名 n FROM 表名;
```

【例 6.3】从 fruits 表中获取 f_name 和 f_price 两列数据，SQL 语句如下：

```
SELECT f_name, f_price FROM fruits;
```

该语句使用 SELECT 声明从 fruits 表中获取名称为 f_name 和 f_price 两个字段下的所有水果名称和价格，两个字段之间用逗号分隔开，查询结果如下：


```
SQL> SELECT f_name, f_price FROM fruits;
```

```
F_NAME      F_PRICE
```

```
-----
apple        5.20
apricot       2.20
blackberry    10.20
berry         7.60
xxxx          3.60
orange        11.20
melon         8.20
cherry        3.20
lemon         6.40
mango         15.60
xbabay        2.60
xxtt          11.60
coconut       9.20
banana        10.30
grape         5.30
xbababa       3.60
```

输出结果显示了 fruits 表中 f_name 和 f_price 两个字段下的所有数据。

提示

Oracle 中的 SQL 语句是不区分大小写的，因此 SELECT 和 select 作用是相同的，但是，许多开发人员习惯将关键字使用大写，而数据列和表名使用小写，读者也应该养成一个良好的编程习惯，这样写出来的代码更容易阅读和维护。

6.2.3 查询指定记录

数据库中包含大量的数据，根据特殊要求，可能只需要查询表中的指定数据，即对数据进行过滤。在 SELECT 语句中，通过 WHERE 子句可以对数据进行过滤，语法格式为：

```
SELECT 字段名1, 字段名2, ..., 字段名 n
```

```
FROM 表名
```

```
WHERE 查询条件
```

在 WHERE 子句中，Oracle 提供了一系列的条件判断符，查询结果如表 6-1 所示。

表 6-1 WHERE 条件判断符

操作符	说明
=	相等
<> , !=	不相等
<	小于
<=	小于或者等于
>	大于
>=	大于或者等于
BETWEEN	位于两值之间

【例 6.4】查询价格为 10.2 元的水果的名称，SQL 语句如下：

```
SELECT f_name, f_price
FROM fruits
WHERE f_price = 10.2;
```

查询结果如下。该语句使用 SELECT 声明从 fruits 表中获取价格等于 10.2 的水果的数据，从查询结果可以看到，价格是 10.2 的水果的名称是 blackberry，其他的均不满足查询条件。

```
F_NAME    F_PRICE
-----
blackberry 10.2
```

本例采用了简单的相等过滤，查询一个指定列 f_price 具有值 10.20。
相等还可以用来比较字符串，如下：

【例 6.5】查找名称为“apple”的水果的价格，SQL 语句如下：

```
SELECT f_name, f_price
FROM fruits
WHERE f_name = 'apple';
```

执行结果如下。该语句使用 SELECT 声明从 fruits 表中获取名称为“apple”的水果的价格，从查询结果可以看到只有名称为“apple”的行被返回，其他的均不满足查询条件。

```
F_NAME    F_PRICE
-----
apple      5.20
```

【例 6.6】查询价格小于 10 的水果的名称，SQL 语句如下：

```
SELECT f_name, f_price
```

```
FROM fruits
WHERE f_price < 10;
```

该语句使用 SELECT 声明从 fruits 表中获取价格低于 10 的水果名称, 即 f_price 小于 10 的水果信息被返回, 查询结果如下:

F_NAME	F_PRICE
apple	5.20
apricot	2.20
berry	7.60
xxxx	3.60
melon	8.20
cherry	3.20
lemon	6.40
xbabay	2.60
coconut	9.20
grape	5.30
xbababa	3.60

可以看到, 查询结果中所有记录的 f_price 字段的值均小于 10.00 元, 而大于或等于 10.00 元的记录没有被返回。

6.2.4 带 IN 关键字的查询

IN 操作符用来查询满足指定范围内的条件的记录, 使用 IN 操作符, 将所有检索条件用括号括起来, 检索条件之间用逗号分隔开, 只要满足条件范围内的一个值即为匹配项。

【例 6.7】查询 s_id 为 101 和 102 的记录, SQL 语句如下:

```
SELECT s_id, f_name, f_price
FROM fruits
WHERE s_id IN (101, 102)
ORDER BY f_name;
```

查询结果如下:

S_ID	F_NAME	F_PRICE
101	apple	5.20


```

102 banana      10.30
101 blackberry  10.20
101 cherry      3.20
102 grape       5.30
102 orange      11.20

```

相反的，可以使用关键字 NOT 来检索不在条件范围内的记录。

【例 6.8】查询所有 s_id 不等于 101 也不等于 102 的记录，SQL 语句如下：

```

SELECT s_id, f_name, f_price
FROM fruits
WHERE s_id NOT IN (101, 102)
ORDER BY f_name;

```

查询结果如下：

```

S_ID  F_NAME  F_PRICE
-----
103  apricot    2.20
104  berry      7.60
103  coconut    9.20
104  lemon      6.40
106  mango     15.60
105  melon      8.20
107  xbababa    3.60
105  xbabay     2.60
105  xxtt      11.60
107  xxxx       3.60

```

可以看到，该语句在 IN 关键字前面加上了 NOT 关键字，这使得查询的结果与前面一个的结果正好相反，前面检索了 s_id 等于 101 和 102 的记录，而这里所要求的查询的记录中的 s_id 字段值不等于这两个值中的任何一个。

6.2.5 带 BETWEEN AND 的范围查询

BETWEEN AND 用来查询某个范围内的值，该操作符需要两个参数，即范围的开始值和结束值，如果字段值满足指定的范围查询条件，则这些记录被返回。

【例 6.9】查询价格在 2.00 元到 10.20 元之间的水果名称和价格，SQL 语句如下：

```

SELECT f_name, f_price FROM fruits WHERE f_price BETWEEN 2.00 AND 10.20;

```


查询结果如下:

F_NAME	F_PRICE
-----	-----
apple	5.20
apricot	2.20
blackberry	10.20
berry	7.60
xxxx	3.60
melon	8.20
cherry	3.20
lemon	6.40
xbabay	2.60
coconut	9.20
grape	5.30
xbababa	3.60

可以看到, 返回结果包含了价格从 2.00 元到 10.20 元之间的字段值, 并且端点值 10.20 也包括在返回结果中, 即 **BETWEEN** 匹配范围中所有值, 包括开始值和结束值。

BETWEEN AND 操作符前可以加关键字 **NOT**, 表示指定范围之外的值, 如果字段值不满足指定的范围内的值, 则这些记录被返回。

【例 6.10】 查询价格在 2.00 元到 10.20 元之外的水果名称和价格, SQL 语句如下:

```
SELECT f_name, f_price
FROM fruits
WHERE f_price NOT BETWEEN 2.00 AND 10.20;
```

查询结果如下:

F_NAME	F_PRICE
-----	-----
orange	11.20
mango	15.60
xxtt	11.60
banana	10.30

由结果可以看到, 返回的记录只有 **f_price** 字段大于 10.20 的, 其实, **f_price** 字段小于 2.00 的记录也满足查询条件。因此, 如果表中有 **f_price** 字段小于 2.00 的记录, 也应当作为查询结果。

6.2.6 带 LIKE 的字符匹配查询

在前面的检索操作中，讲述了如何查询多个字段的记录，如何进行比较查询或者是查询一个条件范围内的记录，如果要查找所有的包含字符“ge”的水果名称，该如何查找呢？简单的比较操作在这里已经行不通了，在这里，需要使用通配符进行匹配查找，通过创建查找模式对表中的数据进行比较。执行这个任务的关键字是 LIKE。

通配符是一种在 SQL 的 WHERE 条件子句中拥有特殊意思的字符，SQL 语句中支持多种通配符，可以和 LIKE 一起使用的通配符有“%”和“_”。

1. 百分号通配符 “%”，匹配任意长度的字符，甚至包括零字符

【例 6.11】查找所有以“b”字母开头的水果，SQL 语句如下：

```
SELECT f_id, f_name
FROM fruits
WHERE f_name LIKE 'b%';
```

查询结果如下：

```
F_ID  F_NAME
-----
b1    blackberry
b2    berry
t1    banana
```

该语句查询的结果返回所有以“b”开头的水果的 id 和 name，“%”告诉 Oracle，返回所有以字母“b”开头的记录，不管“b”后面有多少个字符。

在搜索匹配时通配符“%”可以放在不同位置，如【例 6.12】。

【例 6.12】在 fruits 表中，查询 f_name 中包含字母“g”的记录，SQL 语句如下：

```
SELECT f_id, f_name
FROM fruits
WHERE f_name LIKE '%g%';
```

查询结果如下：

```
F_ID  F_NAME
-----
bs1   orange
m1    mango
t2    grape
```

该语句查询字符串中包含字母“g”的水果名称，只要名字中有字符“g”，而前面或后面不管有多少个字符，都满足查询的条件。

【例 6.13】查询以“b”开头，并以“g”结尾的水果的名称，SQL 语句如下：

```
SELECT f_name
FROM fruits
WHERE f_name LIKE 'b%y';
```

查询结果如下：

```
F_NAME
-----
blackberry
berry
```

通过以上查询结果可以看到，“%”用于匹配在指定的位置的任意数目的字符。

2. 下划线通配符 “_”，一次只能匹配任意一个字符

另一个非常有用的通配符是下划线通配符“_”，该通配符的用法和“%”相同，区别是“%”可以匹配多个字符，而“_”只能匹配任意单个字符，如果要匹配多个字符，则需要使用相同个数的“_”。

【例 6.14】在 fruits 表中，查询以字母“y”结尾，且“y”前面只有 4 个字母的记录，SQL 语句如下：

```
SELECT f_id, f_name FROM fruits WHERE f_name LIKE '____y';
```

查询结果如下：

```
F_ID F_NAME
-----
b2   berry
```

从结果可以看到，以“y”结尾且前面只有 4 个字母的记录只有一条。其他记录的 f_name 字段也有以“y”结尾的，但其总的字符串长度不为 5，因此不在返回结果中。

6.2.7 查询空值

创建数据表的时候，设计者可以指定某列中是否可以包含空值（NULL）。空值不同于 0，也不同于空字符串。空值一般表示数据未知、不适用或将在以后添加数据。在 SELECT 语句中使用 IS NULL 子句，可以查询某字段内容为空记录。

下面在数据库中创建数据表 customers，该表中包含了本章需要用到的数据。



```
CREATE TABLE customers
(
  c_id      number(9) NOT NULL,
  c_name    varchar2(50) NOT NULL,
  c_address varchar2(50) NULL,
  c_city    varchar2(50) NULL,
  c_zip     varchar2(10) NULL,
  c_contact varchar2(50) NULL,
  c_email   varchar2(255) NULL,
  PRIMARY KEY (c_id)
);
```

为了演示需要插入数据, 请读者插入执行以下语句。

```
INSERT INTO customers(c_id, c_name, c_address, c_city,
c_zip, c_contact, c_email) VALUES
(10001, 'RedHook', '200 Street ', 'Tianjin',
'300000', 'LiMing', 'LMing@163.com');
INSERT INTO customers(c_id, c_name, c_address, c_city,
c_zip, c_contact, c_email) VALUES
(10002, 'Stars', '333 Fromage Lane',
'Dalian', '116000', 'Zhangbo', 'Jerry@hotmail.com');
INSERT INTO customers(c_id, c_name, c_address, c_city,
c_zip, c_contact, c_email) VALUES
(10003, 'Netbhood', '1 Sunny Place', 'Qingdao', '266000',
'LuoCong', NULL);
INSERT INTO customers(c_id, c_name, c_address, c_city,
c_zip, c_contact, c_email) VALUES
(10004, 'JOTO', '829 Riverside Drive', 'Haikou',
'570000', 'YangShan', 'sam@hotmail.com');
```

查询上述 4 条记录是否成功插入, 查询插入记录的个数, 输入 SQL 语句, 执行结果如下:

```
SELECT COUNT(*) AS cust_num FROM customers;

CUST_NUM
-----
```




【例 6.15】查询 customers 表中 c_email 为空的记录的 c_id、c_name 和 c_email 字段值, SQL 语句如下:

```
SELECT c_id, c_name, c_email FROM customers WHERE c_email IS NULL;
```

查询结果如下:

C_ID	C_NAME	C_EMAIL
------	--------	---------

10003	Netbhood	
-------	----------	--

可以看到, 显示 customers 表中字段 c_email 的值为 NULL 的记录, 满足查询条件。与 IS NULL 相反的是 NOT IS NULL, 该关键字查找字段不为空的记录。

【例 6.16】查询 customers 表中 c_email 不为空的记录的 c_id、c_name 和 c_email 字段值, SQL 语句如下:

```
SELECT c_id, c_name, c_email FROM customers WHERE c_email IS NOT NULL;
```

查询结果如下:

C_ID	C_NAME	C_EMAIL
------	--------	---------

10001	RedHook	LMing@163.com
10002	Stars	Jerry@hotmail.com
10004	JOTO	sam@hotmail.com

可以看到, 查询出来的记录的 c_email 字段都不为空值。

6.2.8 带 AND 的多条件查询

使用 SELECT 查询时, 可以增加查询的限制条件, 这样可以使查询的结果更加精确。Oracle 在 WHERE 子句中使用 AND 操作符限定只有满足所有查询条件的记录才会被返回。可以使用 AND 连接两个甚至多个查询条件, 多个条件表达式之间用 AND 分开。

【例 6.17】在 fruits 表中查询 s_id = 101, 并且 f_price 大于等于 5 的水果价格和名称, SQL 语句如下:

```
SELECT f_id, f_price, f_name FROM fruits WHERE s_id = '101' AND f_price >= 5;
```

查询结果如下:

F_ID	F_PRICE	F_NAME
------	---------	--------



```
-----
a1    5.20   apple
b1    10.20  blackberry
```

前面的语句检索了 s_id=101 的水果供应商所有价格大于等于 5 元的水果名称和价格。WHERE 子句中的条件分为两部分, AND 关键字指示 Oracle 返回所有同时满足两个条件的行。即使是 s_id=101 的水果供应商提供的水果, 如果价格<5, 或者是 s_id 不等于“101”的水果供应商提供的水果, 不管其价格为多少, 均不是要查询的结果。

提示

上述例子的 WHERE 子句中只包含了一个 AND 语句, 把两个过滤条件组合在一起。实际上可以添加多个 AND 过滤条件, 增加条件的同时增加一个 AND 关键字。

【例 6.18】在 fruits 表中查询 s_id = 101 或者 102, 且 f_price 大于 5, 并且 f_name='apple' 的水果价格和名称, SQL 语句如下:

```
SELECT f_id, f_price, f_name FROM fruits
WHERE s_id IN('101', '102') AND f_price > 5 AND f_name = 'apple';
```

查询结果如下:

```
F_ID  F_PRICE  F_NAME
-----
a1    5.20     apple
```

可以看到, 符合查询条件的返回记录只有一条。

6.2.9 带 OR 的多条件查询

与 AND 相反, 在 WHERE 声明中使用 OR 操作符, 表示只需要满足其中一个条件的记录即可返回。OR 也可以连接两个甚至多个查询条件, 多个条件表达式之间用 OR 分开。

【例 6.19】查询 s_id=101 或者 s_id=102 的水果供应商的 f_price 和 f_name, SQL 语句如下:

```
SELECT s_id, f_name, f_price FROM fruits WHERE s_id = 101 OR s_id = 102;
```

查询结果如下:

```
S_ID  F_NAME      F_PRICE
-----
101   apple       5.20
101   blackberry  10.20
102   orange      11.20
101   cherry      3.20
```



```
102 banana      10.30
102 grape       5.30
```

结果显示了 `s_id=101` 和 `s_id=102` 的水果供应商的水果名称和价格, OR 操作符告诉 Oracle, 检索的时候只需要满足其中的一个条件, 不需要全部都满足。如果这里使用 AND 的话, 将检索不到符合条件的数据。

在这里, 也可以使用 IN 操作符实现与 OR 相同的功能, 下面的例子可进行说明。

【例 6.20】查询 `s_id=101` 或者 `s_id=102` 的水果供应商的 `f_price` 和 `f_name` SQL 语句如下:

```
SELECT s_id,f_name, f_price FROM fruits WHERE s_id IN(101,102);
```

查询结果如下:

```
S_ID  F_NAME      F_PRICE
-----
101  apple       5.20
101  blackberry  10.20
102  orange      11.20
101  cherry      3.20
102  banana      10.30
102  grape       5.30
```

在这里可以看到, OR 操作符和 IN 操作符使用后的结果是一样的, 它们可以实现相同的功能。但是, IN 操作符使得检索语句更加简洁明了, 并且 IN 执行的速度要快于 OR。更重要的是, 使用 IN 操作符, 可以执行更加复杂的嵌套查询(后面章节将会讲述)。

提示

OR 可以和 AND 一起使用, 但是在使用时要注意两者的优先级, 由于 AND 的优先级高于 OR, 因此先对 AND 两边的操作数进行操作, 再与 OR 中的操作数结合。

6.2.10 查询结果不重复

从前面的例子可以看到, SELECT 查询返回所有匹配的行。例如, 查询 fruits 表中所有的 `s_id`, 其结果为:

```
S_ID
-----
101
103
101
104
```




```
107
102
105
101
104
106
105
105
103
102
102
107
```

可以看到查询结果返回了 16 条记录，其中有一些重复的 `s_id` 值，有时出于对数据分析的要求，需要消除重复的记录值，如何使查询结果没有重复呢？在 `SELECT` 语句中，可以使用 `DISTINCT` 关键字指示 Oracle 消除重复的记录值。语法格式为：

```
SELECT DISTINCT 字段名 FROM 表名;
```

【例 6.21】 查询 `fruits` 表中 `s_id` 字段的值，返回 `s_id` 字段值且不得重复，SQL 语句如下：

```
SELECT DISTINCT s_id FROM fruits;
```

查询结果如下：

```
S_ID
-----
101
103
104
107
102
105
106
```

可以看到，这次查询结果只返回了 7 条记录的 `s_id` 值，且不再有重复的值，`SELECT DISTINCT s_id` 告诉 Oracle 只返回不同的 `s_id` 行。



6.2.11 对查询结果排序

从前面的查询结果，读者会发现有些字段的值是没有任何顺序的，Oracle 可以通过在 SELECT 语句中使用 ORDER BY 子句，对查询的结果进行排序。

1. 单列排序

例如查询 f_name 字段，查询结果如下：

```
SQL> SELECT f_name FROM fruits;
```

```
F_NAME
```

```
-----
```

```
apple
```

```
apricot
```

```
blackberry
```

```
berry
```

```
xxxx
```

```
orange
```

```
melon
```

```
cherry
```

```
lemon
```

```
mango
```

```
xbabay
```

```
xxtt
```

```
coconut
```

```
banana
```

```
grape
```

```
xbababa
```

可以看到，查询的数据并没有以一种特定的顺序显示，如果没有对它们进行排序，它们将根据插入到数据表中的顺序来显示。

下面使用 ORDER BY 子句对指定的列数据进行排序。

【例 6.22】 查询 fruits 表的 f_name 字段值，并对其进行排序，SQL 语句如下：

```
SQL> SELECT f_name FROM fruits ORDER BY f_name;
```

```
F_NAME
```

```
-----
```

```
apple
```




```
apricot
banana
berry
blackberry
cherry
coconut
grape
lemon
mango
melon
orange
xbababa
xbabay
xxtt
xxxx
```

该语句查询的结果和前面的语句相同，不同的是，通过指定 ORDER BY 子句，Oracle 对查询的 f_name 列的数据，按字母表的顺序进行了升序排序。

2. 多列排序

有时，需要根据多列值进行排序。比如，如果要显示一个学生列表，可能会有多个学生的姓氏是相同的，因此还需要根据学生的名进行排序。对多列数据进行排序，须将需要排序的列之间用逗号隔开。

【例 6.23】查询 fruits 表中的 f_name 和 f_price 字段，先按 f_name 排序，再按 f_price 排序，SQL 语句如下：

```
SELECT f_name, f_price FROM fruits ORDER BY f_name, f_price;
```

查询结果如下：

F_NAME	F_PRICE
apple	5.20
apricot	2.20
banana	10.30
berry	7.60
blackberry	10.20



cherry	3.20
coconut	9.20
grape	5.30
lemon	6.40
mango	15.60
melon	8.20
orange	11.20
xbababa	3.60
xbabay	2.60
xxtt	11.60
xxxx	3.60

提示

在对多列进行排序的时候,首先排序的第一列必须有相同的列值,才会对第二列进行排序。如果第一列数据中所有值都是唯一的,将不再对第二列进行排序。

3. 指定排序方向

默认情况下,查询数据按字母升序进行排序(从 A~Z),但数据的排序并不仅限于此,还可以使用 ORDER BY 对查询结果进行降序排序(从 Z~A),这可以通过关键字 DESC 实现,下面的例子表明了如何进行降序排列。

【例 6.24】查询 fruits 表中的 f_name 和 f_price 字段,对结果按 f_price 降序方式排序,SQL 语句如下:

```
SELECT f_name, f_price FROM fruits ORDER BY f_price DESC;
```

查询结果如下:

F_NAME	F_PRICE
mango	15.60
xxtt	11.60
orange	11.20
banana	10.30
blackberry	10.20
coconut	9.20
melon	8.20
berry	7.60



lemon	6.40
grape	5.30
apple	5.20
xxxx	3.60
xbababa	3.60
cherry	3.20
xbabay	2.60
apricot	2.20

提示

与 DESC 相反的是 ASC（升序排序），ASC 将字段列中的数据，按字母表顺序升序排序。实际上，在排序的时候 ASC 是作为默认的排序方式，所以加不加都可以。

也可以对多列进行不同的顺序排序，如【例 6.25】所示。

【例 6.25】查询 fruits 表，先按 f_price 降序排序，再按 f_name 字段升序排序，SQL 语句如下：

```
SELECT f_price, f_name FROM fruits ORDER BY f_price DESC, f_name;
```

查询结果如下：

F_PRICE	F_NAME
15.60	mango
11.60	xxtt
11.20	orange
10.30	banana
10.20	blackberry
9.20	coconut
8.20	melon
7.60	berry
6.40	lemon
5.30	grape
5.20	apple
3.60	xbababa
3.60	xxxx
3.20	cherry



```
2.60      xbabay
2.20      apricot
```

由结果可以看出，DESC 排序方式只应用到直接位于其前面的字段上。

提示

DESC 关键字只对其前面的列进行降序排列，在这里只对 `f_price` 排序，而并没有对 `f_name` 进行排序，因此，`f_price` 按降序排序，而 `f_name` 仍按升序排序。如果要对多列都进行降序排序，必须在每一列的列名后面加 DESC 关键字。

6.2.12 分组查询

分组查询是对数据按照某个或多个字段进行分组，Oracle 中使用 GROUP BY 关键字对数据进行分组，基本语法形式为：

```
[GROUP BY 字段] [HAVING <条件表达式>]
```

其中，“字段”为进行分组时所依据的列名称；“HAVING <条件表达式>”指定满足表表达式限定条件的结果将被显示。

1. 创建分组

GROUP BY 关键字通常和集合函数一起使用，如 MAX()、MIN()、COUNT()、SUM()、AVG()。例如，要返回每个水果供应商提供的水果种类，这时就要在分组过程中用到 COUNT() 函数，把数据分为多个逻辑组，并对每个组进行集合计算。

【例 6.26】根据 `s_id` 对 `fruits` 表中的数据进行分组，SQL 语句如下：

```
SELECT s_id, COUNT(*) AS Total FROM fruits GROUP BY s_id;
```

查询结果如下：

s_id	Total
101	3
102	3
103	2
104	2
105	3
106	1
107	2

查询结果显示，`s_id` 表示供应商的 ID，`Total` 字段使用 COUNT() 函数计算得出，GROUP BY 子句按照 `s_id` 排序并对数据分组，可以看到 ID 为 101、102、105 的供应商分别提供 3 种水果，ID 为 103、104、107 的供应商分别提供 2 种水果，ID 为 106 的供应商只提供 1 种水果。



如果要查看每个供应商提供的水果的种类的名称,该怎么办呢? Oracle 可以在 GROUP BY 子句中使用 LISTAGG() 函数, 将每个分组中各个字段的值显示出来。

【例 6.27】根据 s_id 对 fruits 表中的数据进行分组, 将每个供应商的水果名称显示出来, SQL 语句如下:

```
SELECT s_id, LISTAGG(f_name, ',') within group (order by s_id ) AS Names FROM
fruits GROUP BY s_id;
```

查询结果如下:

S_ID	NAMES
101	apple,blackberry,cherry
102	grape,banana,orange
103	apricot,coconut
104	lemon,berry
105	xbabay,xxtt,melon
106	mango
107	xxxx,xbababa

由结果可以看到, LISTAGG() 函数将每个分组中的名称显示出来了, 其名称的个数与 COUNT() 函数计算出来的相同。

2. 使用 HAVING 过滤分组

GROUP BY 可以和 HAVING 一起限定显示记录所需满足的条件, 只有满足条件的分组才会被显示。

【例 6.28】根据 s_id 对 fruits 表中的数据进行分组, 并显示水果种类大于 1 的分组信息, SQL 语句如下:

```
SELECT s_id, LISTAGG(f_name, ',') within group (order by s_id ) AS Names
FROM fruits
GROUP BY s_id HAVING COUNT(f_name) > 1;
```

查询结果如下:

S_ID	NAMES
101	apple,blackberry,cherry
102	grape,banana,orange



```
103  apricot,coconut
104  lemon,berry
105  xbabay,xxtt,melon
107  xxxx,xbababa
```

由结果可以看到, ID 为 101、102、103、104、105、107 的供应商提供的水果种类大于 1, 满足 HAVING 子句条件, 因此出现在返回结果中; 而 ID 为 106 的供应商的水果种类等于 1, 不满足限定条件, 因此不在返回结果中。

提示

HAVING 关键字与 WHERE 关键字都可以用来过滤数据, 两者有什么区别呢? 其中重要的一点是, HAVING 在数据分组之后进行过滤来选择分组, 而 WHERE 在分组之前用来选择记录。另外, WHERE 排除的记录不再包括在分组中。

3. 在 GROUP BY 子句中使用 ROLLUP

使用 ROLLUP 关键字之后, 在所有查询出的分组记录之后增加一条记录, 该记录计算查询出的所有记录的总和, 即统计记录数量。

【例 6.29】根据 s_id 对 fruits 表中的数据进行分组, 并显示记录数量, SQL 语句如下:

```
SELECT s_id, COUNT(*) AS Total
FROM fruits
GROUP BY ROLLUP(s_id) ;
```

查询结果如下:

S_ID	TOTAL
101	3
102	3
103	2
104	2
105	3
106	1
107	2
	16

由结果可以看到, 通过 GROUP BY 分组之后, 在显示结果的最后面新添加了一行, 该行 Total 列的值正好是上面所有数值之和。



4. 多字段分组

使用 GROUP BY 可以对多个字段进行分组，GROUP BY 关键字后面跟需要分组的字段，Oracle 根据多字段的值来进行层次分组，分组层次从左到右，即先按第 1 个字段分组，然后在第 1 个字段值相同的记录中，再根据第 2 个字段的值进行分组，……，依此类推。

【例 6.30】根据 s_id 和 f_name 字段对 fruits 表中的数据进行分组，SQL 语句如下，

```
SQL> SELECT f_id, s_id, f_name, f_price FROM fruits group by s_id, f_name;
```

查询结果如下：

F_ID	S_ID	F_NAME	F_PRICE
a1	101	apple	5.20
b1	101	blackberry	10.20
c0	101	cherry	3.20
t1	102	banana	10.30
t2	102	grape	5.30
bs1	102	orange	11.20
a2	103	apricot	2.20
o2	103	coconut	9.20
b2	104	berry	7.60
l2	104	lemon	6.40
bs2	105	melon	8.20
m2	105	xbabay	2.60
m3	105	xxtt	11.60
m1	106	mango	15.60
t4	107	xbababa	3.60
b5	107	xxxx	3.60

由结果可以看到，查询记录先按照 s_id 进行分组，再对 f_name 字段按不同的取值进行分组。

5. GROUP BY 和 ORDER BY 一起使用

某些情况下需要对分组进行排序，在前面的介绍中，ORDER BY 用来对查询的记录排序，如果和 GROUP BY 一起使用可以完成对分组的排序。

为了演示效果，首先创建数据表，SQL 语句如下：

```
CREATE TABLE orderitems
```



```
(  
    o_num      number(9)          NOT NULL,  
    o_item     number(6)          NOT NULL,  
    f_id       varchar2(10)       NOT NULL,  
    quantity   number(6)          NOT NULL,  
    item_price number(8,2) NOT NULL,  
    PRIMARY KEY (o_num,o_item)  
) ;
```

然后插入演示数据，SQL 语句如下：

```
INSERT INTO orderitems(o_num, o_item, f_id, quantity, item_price)  
SELECT 30001, 1, 'a1', 10, 5.2 from dual  
Union all  
SELECT 30001, 2, 'b2', 3, 7.6 from dual  
Union all  
SELECT 30001, 3, 'bs1', 5, 11.2 from dual  
Union all  
SELECT 30001, 4, 'bs2', 15, 9.2 from dual  
Union all  
SELECT 30002, 1, 'b3', 2, 20.0 from dual  
Union all  
SELECT 30003, 1, 'c0', 100, 10 from dual  
Union all  
SELECT 30004, 1, 'o2', 50, 2.50 from dual  
Union all  
SELECT 30005, 1, 'c0', 5, 10 from dual  
Union all  
SELECT 30005, 2, 'b1', 10, 8.99 from dual  
Union all  
SELECT 30005, 3, 'a2', 10, 2.2 from dual  
Union all  
SELECT 30005, 4, 'm1', 5, 14.99 from dual;
```

【例 6.31】 查询订单价格大于 100 的订单号和总订单价格，SQL 语句如下：



```
SELECT o_num, SUM(quantity * item_price) AS orderTotal
FROM orderitems
GROUP BY o_num
HAVING SUM(quantity*item_price) > 100;
```

查询结果如下：

O_NUM	ORDERTOTAL
30001	268.8
30003	1000
30004	125
30005	236.85

可以看到，返回的结果中 `orderTotal` 列的总订单价格并没有按照一定顺序显示，接下来使用 `ORDER BY` 关键字按总订单价格排序显示结果，SQL 语句如下：

```
SELECT o_num, SUM(quantity * item_price) AS orderTotal
FROM orderitems
GROUP BY o_num
HAVING SUM(quantity*item_price) > 100
ORDER BY orderTotal;
```

查询结果如下：

O_NUM	ORDERTOTAL
30004	125.00
30005	236.85
30001	268.80
30003	1000.00

由结果可以看到，`GROUP BY` 子句按订单号对数据进行分组，`SUM()`函数便可以返回总的订单价格，`HAVING` 子句对分组数据进行过滤，使得只返回总价格大于 100 的订单，最后使用 `ORDER BY` 子句排序输出。

提示

当使用 `ROLLUP` 时，不能同时使用 `ORDER BY` 子句进行结果排序，即 `ROLLUP` 和 `ORDER BY` 是互相排斥的。



6.2.13 使用 ROWNUM 限制查询结果的数量

SELECT 返回所有匹配的行，有可能是表中所有的行，如果仅需要返回第一行或者前几行，则可以使用 ROWNUM 来限制。

【例 6.32】显示 fruits 表查询结果的前 4 行，SQL 语句如下：

```
SELECT * FROM fruits where ROWNUM< 5;
```

查询结果如下：

f_id	s_id	f_name	f_price
a1	101	apple	5.20
a2	103	apricot	2.20
b1	101	blackberry	10.20
b2	104	berry	7.60

由结果可以看到，显示结果从第一行开始，“行数”为小于 5 行，因此返回的结果为表中的前 4 行记录。

提示

使用 ROWNUM 时，只支持 <、<= 和 != 符号，不支持 >、>=、= 和 BETWEEN...AND 符号。

6.3 使用聚合函数查询

有时候并不需要返回实际表中的数据，而只是对数据进行总结。Oracle 提供一些查询功能，可以对获取的数据进行分析和报告。这些函数的功能有：计算数据表中记录行数的总数、计算某个字段列下数据的总和，以及计算表中某个字段下的最大值、最小值或者平均值。本节将介绍这些函数以及如何使用它们。这些聚合函数的名称和作用如表 6-2 所示。

表 6-2 Oracle 聚合函数

函数	作用
AVG()	返回某列的平均值
COUNT()	返回某列的行数
MAX()	返回某列的最大值
MIN()	返回某列的最小值
SUM()	返回某列值的和

接下来，将详细介绍各个函数的使用方法。

6.3.1 COUNT()函数

COUNT()函数统计数据表中包含的记录行的总数，或者根据查询结果返回列中包含的数据行数。其使用方法有两种：

- COUNT(*) 计算表中总的行数，不管某列是否有空值。
- COUNT(字段名)计算指定列下总的行数，计算时将忽略空值的行。

【例 6.33】查询 customers 表中总的行数，SQL 语句如下：

```
SQL> SELECT COUNT(*) AS cust_num FROM customers;

CUST_NUM
-----
         4
```

由查询结果可以看到，COUNT(*)返回 customers 表中记录的总行数，不管其值是什么。返回的总数的名称为 cust_num。

【例 6.34】查询 customers 表中有电子邮箱的顾客的总数，SQL 语句如下：

```
SQL> SELECT COUNT(c_email) AS email_num FROM customers;

EMAIL_NUM
-----
         3
```

由查询结果可以看到，表中 4 个 customer 只有 3 个有 email，customer 的 email 为空值 NULL 的记录没有被 COUNT()函数计算。

提示

两个例子中不同的数值，说明了两种方式在计算总数的时候对待 NULL 值的方式不同。即指定列的值为空的行被 COUNT()函数忽略，但是如果不指定列，而在 COUNT()函数中使用星号“*”，则所有记录都不忽略。

前面介绍分组查询的时候，介绍了 COUNT()函数与 GROUP BY 关键字一起使用，用来计算不同分组中的记录总数。

【例 6.35】在 orderitems 表中，使用 COUNT()函数统计不同订单号中订购的水果种类，SQL 语句如下：

```
SQL> SELECT o_num, COUNT(f_id) FROM orderitems GROUP BY o_num;

O_NUM  COUNT(F_ID)
-----
30001      4
30002      1
```

30003	1
30004	1
30005	4

由查询结果可以看到，GROUP BY 关键字先按照订单号进行分组，然后计算每个分组中的总记录数。

6.3.2 SUM()函数

SUM()是一个求总和的函数，返回指定列值的总和。

【例 6.36】在 orderitems 表中查询 30005 号订单一共购买的水果总量，SQL 语句如下：

```
SQL> SELECT SUM(quantity) AS items_total FROM orderitems WHERE o_num = 30005;
ITEMS_TOTAL
-----
30
```

由查询结果可以看到，SUM(quantity)函数返回订单中所有水果数量之和，WHERE 子句指定查询的订单号为 30005。

SUM()可以与 GROUP BY 一起使用，用来计算每个分组的总和。

【例 6.37】在 orderitems 表中，使用 SUM()函数统计不同订单号中订购的水果总量，SQL 语句如下：

```
SQL> SELECT o_num, SUM(quantity) AS items_total FROM orderitems GROUP BY o_num;

O_NUM  ITEMS_TOTAL
-----
30001   33
30002    2
30003   100
30004   50
30005   30
```

由查询结果可以看到，GROUP BY 按照订单号 o_num 进行分组，SUM()函数计算每个分组中订购的水果的总量。

提示

SUM()函数在计算时，忽略列值为 NULL 的行。

6.3.3 AVG()函数

AVG()函数通过计算返回的行数和每一列数据的和，求得指定列数据的平均值。

【例 6.38】在 fruits 表中，查询 s_id=103 的供应商的水果价格的平均值，SQL 语句如下：

```
SQL> SELECT AVG(ALL f_price) AS avg_price FROM fruits WHERE s_id = 103;
```

AVG_PRICE
5.7

该例中，查询语句增加了一个 WHERE 子句，并且添加了查询过滤条件，只查询 s_id = 103 的记录中的 f_price。因此，通过 AVG()函数计算的结果只是指定的供应商的水果价格的平均值，而不是市场上所有水果的价格的平均值。

AVG()可以与 GROUP BY 一起使用，用来计算每个分组的平均值。

【例 6.39】在 fruits 表中，查询每一个供应商的水果价格的平均值，SQL 语句如下：

```
SQL> SELECT s_id,AVG( ALL f_price) AS avg_price FROM fruits GROUP BY s_id;
```

S_ID	AVG_PRICE
101	6.2
102	8.933333
103	5.7
104	7
105	7.466667
106	15.6
107	3.6

由结果可以看到，GROUP BY 关键字根据 s_id 字段对记录进行分组，然后计算出每个分组的平均值，这种分组求平均值的方法非常有用，例如求不同班级学生成绩的平均值，求不同部门工人的平均工资，求各地的年平均气温等。



AVG()函数使用时，其参数为要计算的列名称，如果要得到多个列的多个平均值，则需要 在每一列上使用 AVG()函数。

6.3.4 MAX()函数

MAX()返回指定列中的最大值。

【例 6.40】在 fruits 表中查找市场上价格最高的水果，SQL 语句如下：

```
SQL>SELECT MAX(f_price) AS max_price FROM fruits;
```

```
MAX_PRICE
```

```
-----
```

```
15.6
```

由结果可以看到，MAX()函数查询出了 f_price 字段的最大值 15.6。

MAX()也可以和 GROUP BY 关键字一起使用，求每个分组中的最大值。

【例 6.41】在 fruits 表中查找不同供应商提供的价格最高的水果，SQL 语句如下：

```
SQL> SELECT s_id, MAX(f_price) AS max_price FROM fruits GROUP BY s_id;
```

```
S_ID  MAX_PRICE
```

```
-----
```

```
101    10.2
```

```
102    11.2
```

```
103     9.2
```

```
104     7.6
```

```
105    11.6
```

```
106    15.6
```

```
107     3.6
```

由结果可以看到，GROUP BY 关键字根据 s_id 字段对记录进行分组，然后计算出每个分组中的最大值。

MAX()函数不仅适用于查找数值类型，也可应用于字符类型。

【例 6.42】在 fruits 表中查找 f_name 的最大值，SQL 语句如下：

```
SQL> SELECT MAX(f_name) FROM fruits;
```

```
MAX(F_NAME)
```

```
-----
```

```
xxxx
```

由结果可以看到，MAX()函数可以对字母进行大小判断，并返回最大的字符或者字符串值。

提示

MAX()函数除了用来找出最大的列值或日期值之外，还可以返回任意列中的最大值，包括返回字符类型的最大值。在对字符类型数据进行比较时，按照字符的 ASCII 码值大小进行比较，从 a~z，a 的 ASCII 码最小，z 的最大。在比较时，先比较第一个字母，如果相等，继续比较下一个字符，一直到两个字符不相等或者字符结束为止。例如，“b”与“t”比较时，“t”为最大值；“bcd”与“bca”比较时，“bcd”为最大值。

6.3.5 MIN()函数

MIN()返回查询列中的最小值。

【例 6.43】在 fruits 表中查找市场上价格最低的水果，SQL 语句如下：

```
SQL>SELECT MIN(f_price) AS min_price FROM fruits;
```

```
MIN_PRICE
-----
2.2
```

由结果可以看到，MIN()函数查询出了 f_price 字段的最小值 2.2。

MIN()也可以和 GROUP BY 关键字一起使用，求出每个分组中的最小值。

【例 6.44】在 fruits 表中查找不同供应商提供的价格最低的水果，SQL 语句如下：

```
SQL> SELECT s_id, MIN(f_price) AS min_price FROM fruits GROUP BY s_id;
```

```
S_ID  MIN_PRICE
-----
101    3.2
102    5.3
103    2.2
104    6.4
105    2.6
106   15.6
107    3.6
```

由结果可以看到，GROUP BY 关键字根据 s_id 字段对记录进行分组，然后计算出每个分组中的最小值。

MIN()函数与 MAX()函数类似，不仅适用于查找数值类型，也可应用于字符类型。

6.4 连接查询

连接是关系数据库模型的主要特点。连接查询是关系数据库中最主要的查询，主要包括内连接、外连接等。通过连接运算符可以实现多个表查询。在关系数据库管理系统中，建立表时各数据之间的关系不必确定，常把一个实体的所有信息存放在一个表中。当查询数据时，通过连接操作查询出存放在多个表中的不同实体的信息。当两个或多个表中存在相同意义的字段时，便可以通过这些字段对不同的表进行连接查询。本节将介绍多表之间的内连接查询、外连接查询以及复合条件连接查询。

6.4.1 内连接查询

内连接（INNER JOIN）使用比较运算符进行表间某（些）列数据的比较操作，并列出这些表中与连接条件相匹配的数据行，组合成新的记录。也就是说，在内连接查询中，只有满足条件的记录才能出现在结果关系中。

为了演示的需要，首先创建数据表 `suppliers`，SQL 语句如下：

```
CREATE TABLE suppliers
(
    s_id      number(9)      NOT NULL,
    s_name    varchar2(50) NOT NULL,
    s_city    varchar2 (50) NULL,
    s_zip     varchar2 (10) NULL,
    s_call    varchar2 (50) NOT NULL,
    PRIMARY KEY (s_id)
) ;
```

插入需要演示的数据，SQL 语句如下：

```
INSERT INTO suppliers(s_id, s_name,s_city, s_zip, s_call)
SELECT 101,'FastFruit Inc.','Tianjin','300000','48075' from dual
Union all
SELECT 102,'LT Supplies','Chongqing','400000','44333' from dual
Union all
SELECT 103,'ACME','Shanghai','200000','90046' from dual
Union all
SELECT 104,'FNK Inc.','Zhongshan','528437','11111' from dual
Union all
```



```

SELECT 105,'Good Set','Taiyuan','030000', '2222' from dual
Union all
SELECT 106,'Just Eat Ours','Beijing','010', '45678' from dual
Union all
SELECT 107,'DK Inc.','Zhengzhou','450000', '33332' from dual;

```

【例 6.45】在 fruits 表和 suppliers 表之间使用内连接查询。

查询之前，查看两个表的结构：

```
SQL> DESC fruits;
```

名称	空值	类型
F_ID	NOT NULL	VARCHAR2(10)
S_ID	NOT NULL	NUMBER(6)
F_NAME	NOT NULL	VARCHAR2(255)
F_PRICE	NOT NULL	NUMBER(8,2)

```
SQL> DESC suppliers;
```

名称	空值	类型
S_ID	NOT NULL	NUMBER(9)
S_NAME	NOT NULL	VARCHAR2(50)
S_CITY		VARCHAR2(50)
S_ZIP		VARCHAR2(10)
S_CALL	NOT NULL	VARCHAR2(50)

由结果可以看到，fruits 表和 suppliers 表中都有相同数据类型的字段 s_id，两个表通过 s_id 字段建立联系。接下来，从 fruits 表中查询 f_name、f_price 字段，从 suppliers 表中查询 s_id、s_name，SQL 语句如下：

```

SQL> SELECT suppliers.s_id, s_name, f_name, f_price FROM fruits ,suppliers
WHERE fruits.s_id = suppliers.s_id;

```


S_ID	S_NAME	F_NAME	F_PRICE
101	FastFruit Inc.	apple	5.20
103	ACME	apricot	2.20
101	FastFruit Inc.	blackberry	10.20
104	FNK Inc.	berry	7.60
107	DK Inc.	xxxx	3.60
102	LT Supplies	orange	11.20
105	Good Set	melon	8.20
101	FastFruit Inc.	cherry	3.20
104	FNK Inc.	lemon	6.40
106	Just Eat Ours	mango	15.60
105	Good Set	xbabay	2.60
105	Good Set	xxtt	11.60
103	ACME	coconut	9.20
102	LT Supplies	banana	10.30
102	LT Supplies	grape	5.30
107	DK Inc.	xbababa	3.60

在这里，SELECT 语句与前面所介绍的一个最大的差别是：SELECT 后面指定的列分别属于两个不同的表，f_name, f_price 在表 fruits 中，而另外两个字段在表 suppliers 中；同时 FROM 子句列出了两个表 fruits 和 suppliers。WHERE 子句在这里作为过滤条件，指明只有两个表中的 s_id 字段值相等的时候才符合连接查询的条件。由返回的结果可以看到，显示的记录是由两个表中的不同列值组成的新记录。

提示

因为 fruits 表和 suppliers 表中有相同的字段 s_id，因此在比较的时候，需要完全限定表名（格式为“表名.列名”），如果只给出 s_id，Oracle 将不知道指的是哪一个，并返回错误信息。

下面的内连接查询语句返回与前面完全相同的结果。

【例 6.46】在 fruits 表和 suppliers 表之间，使用 INNER JOIN 语法进行内连接查询，SQL 语句如下：

```
SQL> SELECT suppliers.s_id, s_name, f_name, f_price FROM fruits INNER JOIN
suppliers ON fruits.s_id = suppliers.s_id;
```


S_ID	S_NAME	F_NAME	F_PRICE
101	FastFruit Inc.	apple	5.20
103	ACME	apricot	2.20
101	FastFruit Inc.	blackberry	10.20
104	FNK Inc.	berry	7.60
107	DK Inc.	xxxx	3.60
102	LT Supplies	orange	11.20
105	Good Set	melon	8.20
101	FastFruit Inc.	cherry	3.20
104	FNK Inc.	lemon	6.40
106	Just Eat Ours	mango	15.60
105	Good Set	xbabay	2.60
105	Good Set	xxtt	11.60
103	ACME	coconut	9.20
102	LT Supplies	banana	10.30
102	LT Supplies	grape	5.30
107	DK Inc.	xbababa	3.60

在这里的查询语句中，两个表之间的关系通过 INNER JOIN 指定。使用这种语法的时候，连接的条件使用 ON 子句给出而不是 WHERE，ON 和 WHERE 后面指定的条件相同。

提示

使用 WHERE 子句定义连接条件比较简单明了，而 INNER JOIN 语法是 ANSI SQL 的标准规范，使用 INNER JOIN 连接语法能够确保不会忘记连接条件，而且 WHERE 子句在某些时候会影响查询的性能。

如果在一个连接查询中，涉及的两个表都是同一个表，这种查询称为自连接查询。自连接是一种特殊的内连接，它是指相互连接的表在物理上为同一张表，但可以在逻辑上分为两张表。

【例 6.47】 查询供应 f_id='a1' 水果的供应商提供的其他水果种类，SQL 语句如下：

```
SQL> SELECT f1.f_id, f1.f_name FROM fruits f1, fruits f2 WHERE f1.s_id =
f2.s_id AND f2.f_id = 'a1';
```

F_ID	F_NAME
a1	apple


```
b1    blackberry
c0    cherry
```

此处查询的两个表是相同的表，为了防止产生二义性，对表使用了别名，`fruits` 表第 1 次出现的别名为 `f1`，第 2 次出现的别名为 `f2`，使用 `SELECT` 语句返回列时明确指出返回以 `f1` 为前缀的列的全名，`WHERE` 连接两个表，并按照第 2 个表的 `f_id` 对数据进行过滤，返回所需数据。

6.4.2 外连接查询

连接查询将查询多个表中相关联的行，内连接时，返回查询结果集合中的仅是符合查询条件和连接条件的行。但有时候需要包含没有关联的行中的数据，即返回查询结果集合中的不仅包含符合连接条件的行，而且还包括左表（左外连接或左连接）、右表（右外连接或右连接）或两个边接表（全外连接）中的所有数据行。外连接分为左外连接和右外连接。

- `LEFT JOIN`（左连接）：返回包括左表中的所有记录和右表中连接字段相等的记录。
- `RIGHT JOIN`（右连接）：返回包括右表中的所有记录和右表中连接字段相等的记录。

1. `LEFT JOIN` 左连接

左连接的结果包括 `LEFT OUTER` 子句中指定的左表的所有行，而不仅仅是连接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果行中，右表的所有选择列表列均为空值。

首先创建表 `orders`，SQL 语句如下：

```
CREATE TABLE orders
(
    o_num number(9) NOT NULL,
    o_date date NOT NULL,
    c_id number(9) NOT NULL,
    PRIMARY KEY (o_num)
);
```

插入需要演示的数据，SQL 语句如下：

```
INSERT INTO orders(o_num, o_date, c_id)VALUES(30001, '01-9月-2008', 10001);
INSERT INTO orders(o_num, o_date, c_id)VALUES (30002, '12-9月-2008', 10003);
INSERT INTO orders(o_num, o_date, c_id)VALUES (30003, '30-9月-2008', 10004);
INSERT INTO orders(o_num, o_date, c_id)VALUES (30004, '03-10月-2008', 10005);
INSERT INTO orders(o_num, o_date, c_id)VALUES (30005, '08-10月-2008', 10001);
```


【例 6.48】在 customers 表和 orders 表中，查询所有客户，包括没有订单的客户，SQL 语句如下：

```
SQL> SELECT customers.c_id, orders.o_num FROM customers LEFT OUTER JOIN orders
ON customers.c_id = orders.c_id;
```

C_ID	O_NUM
10001	30001
10001	30005
10002	
10003	30002
10004	30003

结果显示了 5 条记录，ID 等于 10002 的客户目前并没有下订单，所以对应的 orders 表中并没有该客户的订单信息，所以该条记录只取出了 customers 表中相应的值，而从 orders 表中取出的值为空值。

2. RIGHT JOIN 右连接

右连接是左连接的反向连接，将返回右表的所有行。如果右表的某行在左表中没有匹配行，左表将返回空值。

【例 6.49】在 customers 表和 orders 表中，查询所有订单，包括没有客户的订单，SQL 语句如下：

```
SQL> SELECT customers.c_id, orders.o_num FROM customers RIGHT OUTER JOIN orders
ON customers.c_id = orders.c_id;
```

C_ID	O_NUM
10001	30001
10003	30002
10004	30003
	30004
10001	30005

结果显示了 5 条记录，订单号等于 30004 的订单的客户可能由于某种原因取消了该订单，对应的 customers 表中并没有该客户的信息，所以该条记录只取出了 orders 表中相应的值，而从 customers 表中取出的值为空值 NULL。

6.4.3 复合条件连接查询

复合条件连接查询是在连接查询的过程中,通过添加过滤条件,限制查询的结果,使查询的结果更加准确。

【例 6.50】在 customers 表和 orders 表中,使用 INNER JOIN 语法查询 customers 表中 ID 为 10001 的客户的订单信息,SQL 语句如下:

```
SQL> SELECT customers.c_id, orders.o_num FROM customers INNER JOIN orders ON
customers.c_id = orders.c_id AND customers.c_id = 10001;
```

C_ID	O_NUM
10001	30001
10001	30005

结果显示,在连接查询时指定查询客户 ID 为 10001 的订单信息,添加了过滤条件之后返回的结果将会变少,因此返回结果只有两条记录。

使用连接查询,并对查询的结果进行排序。

【例 6.51】在 fruits 表和 suppliers 表之间,使用 INNER JOIN 语法进行内连接查询,并对查询结果排序,SQL 语句如下:

```
SQL> SELECT suppliers.s_id, s_name, f_name, f_price FROM fruits INNER JOIN
suppliers ON fruits.s_id = suppliers.s_id
```

```
ORDER BY fruits.s_id;
```

S_ID	S_NAME	F_NAME	F_PRICE
101	FastFruit Inc.	apple	5.20
101	FastFruit Inc.	blackberry	10.20
101	FastFruit Inc.	cherry	3.20
102	LT Supplies	grape	5.30
102	LT Supplies	banana	10.30
102	LT Supplies	orange	11.20
103	ACME	apricot	2.20
103	ACME	coconut	9.20
104	FNK Inc.	lemon	6.40
104	FNK Inc.	berry	7.60
105	Good Set	xbabay	2.60
105	Good Set	xxtt	11.60

105	Good Set	melon	8.20
106	Just Eat Ours	mango	15.60
107	DK Inc.	xxxx	3.60
107	DK Inc.	xbababa	3.60

由结果可以看到，内连接查询的结果按照 fruits.s_id 字段进行了升序排序。

6.5 子查询

子查询指一个查询语句嵌套在另一个查询语句内部的查询，这个特性从 Oracle 4.1 开始引入。在 SELECT 子句中先计算子查询，子查询结果作为外层另一个查询的过滤条件，查询可以基于一个表或者多个表。子查询中常用的操作符有 ANY (SOME)、ALL、IN、EXISTS。子查询可以添加到 SELECT、UPDATE 和 DELETE 语句中，而且可以进行多层嵌套。子查询中也可以使用比较运算符，如 “<”、“<=”、“>”、“>=” 和 “!=” 等。本节将介绍如何在 SELECT 语句中嵌套子查询。

6.5.1 带 ANY、SOME 关键字的子查询

ANY 和 SOME 关键字是同义词，表示满足其中任一条件，它们允许创建一个表达式对子查询的返回值列表进行比较，只要满足内层子查询中的任何一个比较条件，就返回一个结果作为外层查询的条件。

下面定义两个表 tbl1 和 tbl2：

```
CREATE table tbl1 ( num1 INT NOT NULL);
CREATE table tbl2 ( num2 INT NOT NULL);
```

分别向两个表中插入数据：

```
INSERT INTO tbl1 values(1);
INSERT INTO tbl1 values(5);
INSERT INTO tbl1 values(13);
INSERT INTO tbl1 values(27);
INSERT INTO tbl2 values(6);
INSERT INTO tbl2 values(14);
INSERT INTO tbl2 values(11);
INSERT INTO tbl2 values(20);
```

ANY 关键字接在一个比较操作符的后面，表示若与子查询返回的任何值比较为 TRUE，则返回 TRUE。

【例 6.52】返回 tbl2 表的所有 num2 列，然后将 tbl1 中的 num1 的值与之进行比较，只要大于 num2 的任何一个值，即为符合查询条件的结果。

```
SQL> SELECT num1 FROM tbl1 WHERE num1 > ANY (SELECT num2 FROM tbl2);
```

```
NUM1
```

```
-----
```

```
13
```

```
27
```

在子查询中，返回的是 tbl2 表的所有 num2 列结果 (6,14,11,20)，然后将 tbl1 中的 num1 列的值与之进行比较，只要大于 num2 列的任意一个数即为符合条件的结果。

6.5.2 带 ALL 关键字的子查询

ALL 关键字与 ANY 和 SOME 不同，使用 ALL 时需要同时满足所有内层查询的条件。例如，修改前面的例子，用 ALL 关键字替换 ANY。

ALL 关键字接在一个比较操作符的后面，表示若与子查询返回的所有值比较为 TRUE，则返回 TRUE。

【例 6.53】返回 tbl1 表中比 tbl2 表 num2 列所有值都大的值，SQL 语句如下：

```
SQL> SELECT num1 FROM tbl1 WHERE num1 > ALL (SELECT num2 FROM tbl2);
```

```
NUM1
```

```
-----
```

```
27
```

在子查询中，返回的是 tbl2 的所有 num2 列结果 (6,14,11,20)，然后将 tbl1 中的 num1 列的值与之进行比较，大于所有 num2 列值的 num1 值只有 27，因此返回结果为 27。

6.5.3 带 EXISTS 关键字的子查询

EXISTS 关键字后面的参数是一个任意的子查询，系统对子查询进行运算以判断它是否返回行，如果至少返回一行，那么 EXISTS 的结果为 TRUE，此时外层查询语句将进行查询；如果子查询没有返回任何行，那么 EXISTS 返回的结果是 FALSE，此时外层查询语句将不进行查询。

【例 6.54】查询 suppliers 表中是否存在 s_id=107 的供应商，如果存在，则查询 fruits 表中

的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE EXISTS (SELECT s_name FROM suppliers WHERE
s_id = 107);
```

F_ID	S_ID	F_NAME	F_PRICE
a1	101	apple	5.20
a2	103	apricot	2.20
b1	101	blackberry	10.20
b2	104	berry	7.60
b5	107	xxxx	3.60
bs1	102	orange	11.20
bs2	105	melon	8.20
c0	101	cherry	3.20
l2	104	lemon	6.40
m1	106	mango	15.60
m2	105	xbabay	2.60
m3	105	xxtt	11.60
o2	103	coconut	9.20
t1	102	banana	10.30
t2	102	grape	5.30
t4	107	xbababa	3.60

由结果可以看到，内层查询结果表明 `suppliers` 表中存在 `s_id=107` 的记录，因此 `EXISTS` 表达式返回 `TRUE`；外层查询语句接收 `TRUE` 之后对表 `fruits` 进行查询，返回所有的记录。

`EXISTS` 关键字可以和条件表达式一起使用。

【例 6.55】 查询 `suppliers` 表中是否存在 `s_id=107` 的供应商，如果存在，则查询 `fruits` 表中的 `f_price` 大于 10.20 的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE f_price>10.20 AND EXISTS
(SELECT s_name FROM suppliers WHERE s_id = 107);
```

F_ID	S_ID	F_NAME	F_PRICE
bs1	102	orange	11.20
m1	106	mango	15.60

```
m3      105      xxtt      11.60
t1       102     banana     10.30
```

由结果可以看到，内层查询结果表明 `suppliers` 表中存在 `s_id=107` 的记录，因此 `EXISTS` 表达式返回 `TRUE`；外层查询语句接收 `TRUE` 之后根据查询条件 `f_price > 10.20` 对 `fruits` 表进行查询，返回结果为 4 条 `f_price` 大于 10.20 的记录。

`NOT EXISTS` 与 `EXISTS` 使用方法相同，返回的结果相反。子查询如果至少返回一行，那么 `NOT EXISTS` 的结果为 `FALSE`，此时外层查询语句将不进行查询；如果子查询没有返回任何行，那么 `NOT EXISTS` 返回的结果是 `TRUE`，此时外层语句将进行查询。

【例 6.56】查询 `suppliers` 表中是否存在 `s_id=107` 的供应商，如果不存在，则查询 `fruits` 表中的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE NOT EXISTS (SELECT s_name FROM suppliers WHERE
s_id = 107);
```

未选择任何行

查询语句 `SELECT s_name FROM suppliers WHERE s_id = 107`，对 `suppliers` 表进行查询返回了一条记录，`NOT EXISTS` 表达式返回 `FALSE`，外层查询语句接收 `FALSE` 将不再查询 `fruits` 表中的记录。



`EXISTS` 和 `NOT EXISTS` 的结果只取决于是否会返回行，而不取决于这些行的内容，所以这个子查询输入列表通常是无关紧要的。

6.5.4 带 IN 关键字的子查询

`IN` 关键字进行子查询时，内层查询语句仅仅返回一个数据列，这个数据列里的值将提供给外层查询语句进行比较操作。

【例 6.57】在 `orderitems` 表中查询 `f_id` 为 `c0` 的订单号，并根据订单号查询具有订单号的客户 `c_id`，SQL 语句如下：

```
SQL> SELECT c_id FROM orders WHERE o_num IN (SELECT o_num FROM orderitems WHERE
f_id = 'c0');
```

```
C_ID
-----
10004
10001
```

查询结果的 `c_id` 有两个值，分别为 10001 和 10004。上述查询过程可以分步执行，首先内层子查询查出 `orderitems` 表中符合条件的订单号，单独执行内查询，查询结果如下：



```
SQL> SELECT o_num FROM orderitems WHERE f_id = 'c0';
```

```
O_NUM
```

```
-----
```

```
30003
```

```
30005
```

可以看到，符合条件的 o_num 列的值有两个：30003 和 30005，然后执行外层查询，在 orders 表中查询订单号等于 30003 或 30005 的客户 c_id 嵌套子查询语句还可以写为如下形式实现相同的效果：

```
SQL> SELECT c_id FROM orders WHERE o_num IN (30003, 30005);
```

```
C_ID
```

```
-----
```

```
10004
```

```
10001
```

这个例子说明在处理 SELECT 语句的时候，Oracle 实际上执行了两个操作过程，即先执行内层子查询，再执行外层查询，内层子查询的结果作为外部查询的比较条件。

SELECT 语句中可以使用 NOT IN 关键字，其作用与 IN 正好相反。

【例 6.58】与前一个例子类似，但是在 SELECT 语句中使用 NOT IN 关键字，SQL 语句如下：

```
SQL> SELECT c_id FROM orders WHERE o_num NOT IN (SELECT o_num FROM orderitems  
WHERE f_id = 'c0');
```

```
C_ID
```

```
-----
```

```
10001
```

```
10003
```

```
10005
```

这里返回的结果有 3 条记录，由前面可以看到，子查询返回的订单值有两个，即 30003 和 30005，但为什么这里还有值为 10001 的 c_id 呢？这是因为 c_id 等于 10001 的客户的订单不只一个，可以查看订单表 orders 中的记录。

```
SQL> SELECT * FROM orders;
```

```
O_NUM O_DATE C_ID
```




```

-----
30001    01-9月 -08    10001
30004    03-10月-08    10005
30005    08-10月-08    10001
30002    12-9月 -08    10003
30003    30-9月 -08    10004

```

可以看到,虽然排除了订单号为30003和30005的客户c_id,但是o_num为30001与30005的订单都是10001号客户的订单。所以,结果中只是排除了订单号,但是仍然有可能选择同一个客户。

提示

子查询的功能也可以通过连接查询完成,但是子查询使得 Oracle 代码更容易阅读和编写。

6.5.5 带比较运算符的子查询

在前面介绍的带 ANY、ALL 关键字的子查询使用了“>”比较运算符,子查询还可以使用其他的比较运算符,如“<”、“<=”、“=”、“>=”和“!=”等。

【例 6.59】在 suppliers 表中查询 s_city 等于“Tianjin”的供应商 s_id,然后在 fruits 表中查询所有该供应商提供的水果的种类,SQL 语句如下:

```

SELECT s_id, f_name FROM fruits WHERE s_id =
(SELECT s1.s_id FROM suppliers s1 WHERE s1.s_city = 'Tianjin');

```

该嵌套查询首先在 suppliers 表中查找 s_city 等于“Tianjin”的供应商的 s_id,单独执行子查询查看 s_id 的值,执行下面的操作过程:

```
SQL> SELECT s1.s_id FROM suppliers s1 WHERE s1.s_city = 'Tianjin';
```

```
S_ID
```

```
-----
```

```
101
```

然后在外层查询时,在 fruits 表中查找 s_id 等于 101 的供应商提供的水果的种类,查询结果如下:

```
SQL> SELECT s_id, f_name FROM fruits WHERE s_id =
      (SELECT s1.s_id FROM suppliers s1 WHERE s1.s_city = 'Tianjin');
```

```
S_ID  F_NAME
```

```
-----
```

```
101  apple
```




```
101 blackberry
101 cherry
```

结果表明，“Tianjin”地区的供应商提供的水果种类有 3 种，分别为“apple”、“blackberry”、“cherry”。

【例 6.60】在 suppliers 表中查询 s_city 等于“Tianjin”的供应商 s_id，然后在 fruits 表中查询所有非该供应商提供的水果的种类，SQL 语句如下：

```
SQL> SELECT s_id, f_name FROM fruits WHERE s_id <>
(SELECT s1.s_id FROM suppliers s1 WHERE s1.s_city = 'Tianjin');
S_ID  F_NAME
-----
103   apricot
104   berry
107   xxxx
102   orange
105   melon
104   lemon
106   mango
105   xbabay
105   xxtt
103   coconut
102   banana
102   grape
107   xbababa
```

该嵌套查询执行过程与前面相同，在这里使用了不等于“<>”运算符，因此返回的结果和前面正好相反。

6.6 合并查询结果

利用 UNION 关键字，可以给出多条 SELECT 语句，并将它们的结果组合成单个结果集。合并时，两个表对应的列数和数据类型必须相同。各个 SELECT 语句之间使用 UNION 或 UNION ALL 关键字分隔。UNION 不使用关键字 ALL，执行的时候删除重复的记录，所有返回的行都是唯一的；使用关键字 ALL 的作用是不删除重复行也不对结果进行自动排序。基本



语法格式如下：

```
SELECT column,... FROM table1
UNION [ALL]
SELECT column,... FROM table2
```

【例 6.61】查询所有价格小于 9 的水果的信息，查询 s_id 等于 101 和 103 的所有水果的信息，使用 UNION 连接查询结果，SQL 语句如下：

```
SELECT s_id, f_name, f_price
FROM fruits
WHERE f_price < 9.0
UNION
SELECT s_id, f_name, f_price
FROM fruits
WHERE s_id IN(101,103);
```

合并查询结果如下：

S_ID	F_NAME	F_PRICE
101	apple	5.2
103	apricot	2.2
104	berry	7.6
107	xxxx	3.6
105	melon	8.2
101	cherry	3.2
104	lemon	6.4
105	xbabay	2.6
102	grape	5.3
107	xbababa	3.6
101	apple	5.2
103	apricot	2.2
101	blackberry	10.2
101	cherry	3.2
103	coconut	9.2

如前所述，UNION 将多个 SELECT 语句的结果组合成一个结果集合。可以分开查看每个



SELECT 语句的结果：

```
SQL> SELECT s_id, f_name, f_price FROM fruits
      WHERE f_price < 9.0;
```

S_ID	F_NAME	F_PRICE
101	apple	5.2
103	apricot	2.2
104	berry	7.6
107	xxxx	3.6
105	melon	8.2
101	cherry	3.2
104	lemon	6.4
105	xbabay	2.6
102	grape	5.3
107	xbababa	3.6

```
SQL> SELECT s_id, f_name, f_price FROM fruits
      WHERE s_id IN(101,103);
```

S_ID	F_NAME	F_PRICE
101	apple	5.2
103	apricot	2.2
101	blackberry	10.2
101	cherry	3.2
103	coconut	9.2

由分开查询的结果可以看到，第1条 SELECT 语句查询价格小于9的水果，第2条 SELECT 语句查询供应商 101 和 103 提供的水果。使用 UNION 将两条 SELECT 语句分隔开，执行完毕之后把输出结果组合成单个的结果集，并删除重复的记录。

使用 UNION ALL 包含重复的行，在前面的例子中，分开查询时，两个返回结果中有相同的记录。UNION 从查询结果集中自动去除了重复的行，如果要返回所有匹配行，而不进行删除，可以使用 UNION ALL。

【例 6.62】 查询所有价格小于 9 的水果的信息，查询 s_id 等于 101 和 103 的所有水果的信息，使用 UNION ALL 连接查询结果，SQL 语句如下：



```
SELECT s_id, f_name, f_price
FROM fruits
WHERE f_price < 9.0
UNION ALL
SELECT s_id, f_name, f_price
FROM fruits
WHERE s_id IN(101,103);
```

查询结果如下：

S_ID	F_NAME	F_PRICE
101	apple	5.2
103	apricot	2.2
104	berry	7.6
107	xxxx	3.6
105	melon	8.2
101	cherry	3.2
104	lemon	6.4
105	xbabay	2.6
102	grape	5.3
107	xbababa	3.6
101	apple	5.2
103	apricot	2.2
101	blackberry	10.2
101	cherry	3.2
103	coconut	9.2

由结果可以看到，这里总的记录数等于两条 SELECT 语句返回的记录数之和，连接查询结果并没有去除重复的行。

提示

UNION 和 UNION ALL 的区别：使用 UNION ALL 的功能是不删除重复行，加上 ALL 关键字语句执行时所需要的资源少，所以尽可能地使用它，因此知道有重复行但是想保留这些行，确定查询结果中不会有重复数据或者不需要去掉重复数据的时候，应当使用 UNION ALL 以提高查询效率。



6.7 为表和字段取别名

在前面介绍分组查询、集合函数查询和嵌套子查询章节中，读者注意到有的地方使用了为查询结果中的某一列指定一个特定的名字。在内连接查询时，则对相同的表 `fruits` 分别指定两个不同的名字。这里可以为字段或者表取一个别名，在查询时，使用别名替代其指定的内容。本节将介绍如何为字段和表创建别名以及如何使用别名。

6.7.1 为表取别名

当表名字很长或者执行一些特殊查询时，为了方便操作或者需要多次使用相同的表时，可以为表指定别名，用这个别名替代表原来的名称。为表取别名的基本语法格式为：

表名[AS]表别名

其中，“表名”为数据库中存储的数据表的名称，“表别名”为查询时指定的表的新名称，AS 关键字为可选参数。

【例 6.63】为 `orders` 表取别名 `o`，查询 30001 订单的下单日期，SQL 语句如下：

```
SELECT * FROM orders AS o
WHERE o.o_num = 30001;
```

在这里 `orders AS o` 代码表示为 `orders` 表取别名为 `o`，指定过滤条件时直接使用 `o` 代替 `orders`，查询结果如下：

O_NUM	O_DATE	C_ID
30001	01-9月 -08	10001

【例 6.64】为 `customers` 和 `orders` 表分别取别名，并进行连接查询，SQL 语句如下：

```
SQL> SELECT c.c_id, o.o_num
FROM customers c LEFT OUTER JOIN orders o
ON c.c_id = o.c_id;

C_ID O_NUM
-----
10001 30001
10001 30005
10002
```




```
10003 30002
```

```
10004 30003
```

由结果看到, Oracle 可以同时为多个表取别名, 而且表别名可以放在不同的位置, 如 WHERE 子句、SELECT 列表、ON 子句以及 ORDER BY 子句等。

在前面介绍内连接查询时指出自连接是一种特殊的内连接, 在连接查询中的两个表都是同一个表, 其查询语句如下:

```
SQL> SELECT f1.f_id, f1.f_name
      FROM fruits f1, fruits f2
      WHERE f1.s_id = f2.s_id AND f2.f_id = 'a1';
```

```
F_ID F_NAME
```

```
-----
```

```
a1 apple
```

```
b1 blackberry
```

```
c0 cherry
```

在这里, 如果不使用表别名, Oracle 将不知道引用的是哪个 fruits 表实例, 这是表别名的一个非常有用的地方。

提示

在为表取别名时, 要保证不能与数据库中的其他表的名称冲突。

6.7.2 为字段取别名

在本章和前面各章节的例子中可以看到, 在使用 SELECT 语句显示查询结果时, Oracle 会显示每个 SELECT 后面指定的输出列, 在有些情况下, 显示的列的名称会很长或者名称不够直观, Oracle 可以指定列别名, 替换字段或表达式。为字段取别名的基本语法格式为:

```
列名 [AS] 列别名
```

其中, “列名”为表中字段定义的名称, “列别名”为字段新的名称, AS 关键字为可选参数。

【例 6.65】查询 fruits 表, 为 f_name 取别名 fruit_name, f_price 取别名 fruit_price, 为 fruits 表取别名 f1, 查询表中 f_price < 8 的水果的名称, SQL 语句如下:

```
SQL> SELECT f1.f_name AS fruit_name, f1.f_price AS fruit_price
      FROM fruits f1
      WHERE f1.f_price < 8;
```




```
FRUIT_NAME    FRUIT_PRICE
```

```
-----
```

```
apple         5.2
```

```
apricot       2.2
```

```
berry         7.6
```

```
xxxx         3.6
```

```
cherry        3.2
```

```
lemon         6.4
```

```
xbabay        2.6
```

```
grape         5.3
```

```
xbababa       3.6
```

也可以为 SELECT 子句中的计算字段取别名，例如，对使用 COUNT 聚合函数或者 CONCAT 等系统函数执行的结果字段取别名。

【例 6.66】查询 suppliers 表中字段 s_name 和 s_city，使用 CONCAT 函数连接这两个字段值，并取列别名为 suppliers_title。

如果没有对连接后的值取别名，其显示列名称将会不够直观，SQL 语句如下：

```
SQL> SELECT CONCAT(s_name , s_city) FROM suppliers ORDER BY s_name;
```

```
CONCAT(S_NAME,S_CITY)
```

```
-----
```

```
ACMESHanghai
```

```
DK Inc.Zhengzhou
```

```
FNK Inc.Zhongshan
```

```
FastFruit Inc.Tianjin
```

```
Good SetTaiyuan
```

```
Just Eat OursBeijing
```

```
LT SuppliesChongqing
```

由结果可以看到，显示结果的列名称为 SELECT 子句后面的计算字段，实际上计算之后的列是没有名字的，这样的结果让人很不容易理解，如果为字段取一个别名，将会使结果清晰，SQL 语句如下：

```
SQL> SELECT CONCAT(s_name , s_city)
```

```
AS suppliers_title
```

```
FROM suppliers
```




```
ORDER BY s_name;
```

```
SUPPLIERS_TITLE
```

```
-----
```

```
ACMEShanghai
```

```
DK Inc.Zhengzhou
```

```
FNK Inc.Zhongshan
```

```
FastFruit Inc.Tianjin
```

```
Good SetTaiyuan
```

```
Just Eat OursBeijing
```

```
LT SuppliesChongqing
```

由结果可以看到，SELECT 子句计算字段之后增加了 AS suppliers_title，它指示 Oracle 为计算字段创建一个别名 suppliers_title，显示结果为指定的列别名，这样就增强了查询结果的可读性。

提示

表别名只在执行查询的时候使用，并不在返回结果中显示，而列别名定义之后，将返回给客户端显示，显示的结果字段为字段列的别名。

6.8 使用正则表达式查询

正则表达式通常被用来检索或替换那些符合某个模式的文本内容，根据指定的匹配模式匹配文本中符合要求的特殊字符串。例如，从一个文本文件中提取电话号码，查找一篇文章中重复的单词或者替换用户输入的某些敏感词语等，这些地方都可以使用正则表达式。正则表达式强大而且灵活，可以应用于非常复杂的查询。

Oracle 中使用 REGEXP_LIKE() 函数指定正则表达式的字符匹配模式，表 6-3 列出了 REGEXP_LIKE 函数中常用字符匹配列表。

表 6-3 正则表达式常用字符匹配列表

选项	说明	例子	匹配值示例
^	匹配文本的开始字符	“^b” 匹配以字母 b 开头的字符串	book, big, banana, bike
\$	匹配文本的结束字符	“st\$” 匹配以 st 结尾的字符串	test, resist, persist
.	匹配任何单个字符	“b.t” 匹配任何 b 和 t 之间有一个字符	bit, bat, but, bite



(续表)

选项	说明	例子	匹配值示例
*	匹配零个或多个在它前面的字符	“f*n” 匹配字符 n 前面有任意个字符 f	fn, fan, faan, abcn
+	匹配前面的字符 1 次或多次	“ba+ ” 匹配以 b 开头后面紧跟至少有一个 a	Ba, bay, bare, battle
<字符串>	匹配包含指定的字符串的文本	“fa”	Fan, afa, faad
[字符集合]	匹配字符集中的任何一个字符	“[xz]” 匹配 x 或者 z	Dizzy, zebra, x-ray, extra
[^]	匹配不在括号中的任何字符	“[^abc]” 匹配任何不包含 a、b 或 c 的字符串	Desk, fox, f8ke
字符串{n,}	匹配前面的字符串至少 n 次	b{2} 匹配 2 个或更多的 b	Bbb, bbbb, bbbbbbb
字符串{n,m}	匹配前面的字符串至少 n 次，至多 m 次。如果 n 为 0，此参数为可选参数	b{2,4} 匹配最少 2 个，最多 4 个 b	Bb, bbb, bbbb

下文将详细介绍在 Oracle 中如何使用正则表达式。

6.8.1 查询以特定字符或字符串开头的记录

字符 “^” 匹配以特定字符或者字符串开头的文本。

【例 6.67】在 fruits 表中，查询 f_name 字段以字母 “b” 开头的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(f_name , '^b');
```

F_ID	S_ID	F_NAME	F_PRICE
b1	101	blackberry	10.2
b2	104	berry	7.6
t1	102	banana	10.3

fruits 表中有 3 条记录的 f_name 字段值是以字母 “b” 开头，返回结果有 3 条记录。

【例 6.68】在 fruits 表中，查询 f_name 字段以 “be” 开头的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE( f_name , '^be');
```

F_ID	S_ID	F_NAME	F_PRICE
b2	104	berry	7.6



只有 berry 是以“be”开头，所以查询结果中只有 1 条记录。

6.8.2 查询以特定字符或字符串结尾的记录

字符“\$”匹配以特定字符或者字符串结尾的文本。

【例 6.69】在 fruits 表中，查询 f_name 字段以字母“y”结尾的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(f_name , 'y$');
```

F_ID	S_ID	F_NAME	F_PRICE
b1	01	blackberry	10.2
b2	104	berry	7.6
c0	101	cherry	3.2
m2	105	xbabay	2.6

fruits 表中有 4 条记录的 f_name 字段值是以字母“y”结尾，返回结果有 4 条记录。

【例 6.70】在 fruits 表中，查询 f_name 字段以字符串“rry”结尾的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(f_name , 'rry$');
```

F_ID	S_ID	F_NAME	F_PRICE
b1	101	blackberry	10.2
b2	104	berry	7.6
c0	101	cherry	3.2

fruits 表中有 3 条记录的 f_name 字段值是以字符串“rry”结尾，返回结果有 3 条记录。

6.8.3 用符号“.”来替代字符串中的任意一个字符

字符“.”匹配任意一个字符。

【例 6.71】在 fruits 表中，查询 f_name 字段值包含字母“a”与“g”且两个字母之间只有一个字母的记录，SQL 语句如下，

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(f_name , 'a.g');
```

F_ID	S_ID	F_NAME	F_PRICE
------	------	--------	---------



```
bs1 102 orange 11.20
m1 106 mango 15.60
```

查询语句中“a.g”指定匹配字符中要有字母 a 和 g，且两个字母之间包含单个字符，并不限定匹配的字符的位置和所在查询字符串的总长度，因此 orange 和 mango 都符合匹配条件。

6.8.4 使用“*”和“+”来匹配多个字符

星号“*”匹配前面的字符任意多次，包括 0 次。加号“+”匹配前面的字符至少一次。

【例 6.72】在 fruits 表中，查询 f_name 字段值以字母“b”开头，且“b”后面出现字母“a”的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(f_name , '^ba*');

F_ID  S_ID  F_NAME  F_PRICE
-----
b1    101    blackberry  10.20
b2    104    berry      7.60
t1    102    banana     10.30
```

星号“*”可以匹配任意多个字符，berry 中字母 b 后面并没有出现字母 a，但是也满足匹配条件。

【例 6.73】在 fruits 表中，查询 f_name 字段值以字母“b”开头，且“b”后面出现字母“a”至少一次的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(f_name , '^ba+');

F_ID  S_ID  F_NAME  F_PRICE
-----
t1    102    banana     10.30
```

“a+”匹配字母“a”至少一次，只有 banana 满足匹配条件。

6.8.5 匹配指定字符串

正则表达式可以匹配指定字符串，只要这个字符串在查询文本中即可，如要匹配多个字符串，多个字符串之间使用分隔符“|”隔开。

【例 6.74】在 fruits 表中，查询 f_name 字段值包含字符串“on”的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(f_name , 'on');
```




F_ID	S_ID	F_NAME	F_PRICE
bs2	105	melon	8.20
12	104	lemon	6.40
o2	103	coconut	9.20

可以看到, `f_name` 字段的 `melon`、`lemon` 和 `coconut` 三个值中都包含有字符串 “on”, 满足匹配条件。

【例 6.75】 在 `fruits` 表中, 查询 `f_name` 字段值包含字符串 “on” 或者 “ap” 的记录, SQL 语句如下:

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(f_name, 'on|ap');
```

F_ID	S_ID	F_NAME	F_PRICE
a1	101	apple	5.20
a2	103	apricot	2.20
bs2	105	melon	8.20
12	104	lemon	6.40
o2	103	coconut	9.20
t2	102	grape	5.30

可以看到, `f_name` 字段的 `melon`、`lemon` 和 `coconut` 三个值中都包含有字符串 “on”, `apple`、`apricot` 和 `grape` 三个值中包含字符串 “ap”, 满足匹配条件。

提示

之前介绍过, `LIKE` 运算符也可以匹配指定的字符串, 但与 `REGEXP_LIKE` 不同, `LIKE` 匹配的字符串如果在文本中间出现, 则找不到它, 相应的行也不会返回。而 `REGEXP` 在文本内进行匹配, 如果被匹配的字符串在文本中出现, `REGEXP_LIKE` 将会找到它, 相应的行也会被返回。对比结果如 **【例 6.76】** 所示。

【例 6.76】 在 `fruits` 表中, 使用 `LIKE` 运算符查询 `f_name` 字段值为 “on” 的记录, SQL 语句如下:

```
SQL> SELECT * FROM fruits WHERE f_name LIKE 'on';
```

未选择任何行

`f_name` 字段没有值为 “on” 的记录, 返回结果为空。读者可以体会一下两者的区别。



6.8.6 匹配指定字符中的任意一个

方括号 “[] ” 指定一个字符集合，只匹配其中任何一个字符，即为所查找的文本。

【例 6.77】在 fruits 表中，查找 f_name 字段值中包含字母 “o” 或者 “t” 的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(f_name , '[ot]');
```

F_ID	S_ID	F_NAME	F_PRICE
a2	103	apricot	2.20
bs1	102	orange	11.20
bs2	105	melon	8.20
l2	104	lemon	6.40
m1	106	mango	15.60
m3	105	xxtt	11.60
o2	103	coconut	9.20

由查询结果可以看到，所有返回的记录的 f_name 字段值中都包含有字母 “o” 或者 “t”，或者两个都有。

方括号 “[] ” 还可以指定数值集合。

【例 6.78】在 fruits 表，查询 s_id 字段值中包含 4、5 或者 6 的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(s_id , '[456]');
```

F_ID	S_ID	F_NAME	F_PRICE
b2	104	berry	7.60
bs2	105	melon	8.20
l2	104	lemon	6.40
m1	106	mango	15.60
m2	105	xbabay	2.60
m3	105	xxtt	11.60

查询结果中，s_id 字段值中有 3 个数字中的 1 个即为匹配记录字段。

匹配集合 “[456]” 也可以写成 “[4-6]”，即指定集合区间。例如，“[a-z]” 表示集合区间为从 a~z 的字母，“[0-9]” 表示集合区间为从 0~9 的数字。



6.8.7 匹配指定字符以外的字符

“`[^字符集合]`”匹配不在指定集合中的任何字符。

【例 6.79】在 `fruits` 表中，查询 `f_id` 字段值包含字母 `a~e` 和数字 `1~2` 以外的字符的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(f_id , '[^a-e1-2]');
```

F_ID	S_ID	F_NAME	F_PRICE
------	------	--------	---------

b5	107	xxxx	3.6
bs1	102	orange	11.2
bs2	105	melon	8.2
c0	101	cherry	3.2
l2	104	lemon	6.4
m1	106	mango	15.6
m2	105	xbabay	2.6
m3	105	xxtt	11.6
o2	103	coconut	9.2
t1	102	banana	10.3
t2	102	grape	5.3
t4	107	xbababa	3.6

返回记录的 `f_id` 字段值中包含了指定字母和数字以外的值，如 `s`、`m`、`o`、`t` 等，这些字母均不在 `a~e` 与 `1~2` 之间，满足匹配条件。

6.8.8 使用 `{n,}` 或者 `{n,m}` 来指定字符串连续出现的次数

“字符串 `{n,}`”表示至少匹配 `n` 次前面的字符；“字符串 `{n,m}`”表示匹配前面的字符串不少于 `n` 次，不多于 `m` 次。例如，`a{2,}` 表示字母 `a` 连续出现至少 2 次，也可以大于 2 次；`a{2,4}` 表示字母 `a` 连续出现最少 2 次，最多不能超过 4 次。

【例 6.80】在 `fruits` 表中，查询 `f_name` 字段值出现字母 “`x`” 至少 2 次的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(f_name , 'x{2,}');
```

F_ID	S_ID	F_NAME	F_PRICE
------	------	--------	---------



b5	107	xxxx	3.60
m3	105	xttt	11.60

可以看到，f_name 字段的“xxxx”包含 4 个字母“x”，“xttt”包含两个字母“x”，均为满足匹配条件的记录。

【例 6.81】在 fruits 表中，查询 f_name 字段值出现字符串“ba”最少 1 次，最多 3 次的记录，SQL 语句如下：

```
SQL> SELECT * FROM fruits WHERE REGEXP_LIKE(f_name , 'ba{1,3}');
F_ID  S_ID  F_NAME  F_PRICE
-----
m2    105    xbabay    2.6
t1     102    banana    10.3
t4     107    xbababa    3.6
```

可以看到，f_name 字段的“xbabay”中“ba”出现了 2 次，“banana”中出现了 1 次，“xbababa”中出现了 3 次，都为满足匹配条件的记录。

6.9 综合案例——数据表查询操作

SQL 语句可以分为两部分，一部分用来创建数据库对象，另一部分用来操作这些对象，本章详细介绍了操作数据库对象的数据表查询语句。通过本章的介绍，读者可以了解到 SQL 中的查询语言功能的强大，用户可以根据需要灵活使用。本章的综合案例将回顾这些查询语句。

1. 案例目的

根据不同条件对表进行查询操作，掌握数据表的查询语句。employee、dept 表结构以及表中的记录，如表 6-4~表 6-7 所示。

表 6-4 employee 表结构

字段名	字段说明	数据类型	主键	外键	非空	唯一	自增
e_no	员工编号	NUMBER(11)	是	否	是	是	否
e_name	员工姓名	VARCHAR2(50)	否	否	是	否	否
e_gender	员工性别	VARCHAR2(2)	否	否	否	否	否
dept_no	部门编号	NUMBER (11)	否	否	是	否	否
e_job	职位	VARCHAR2(50)	否	否	是	否	否
e_salary	薪水	NUMBER (11)	否	否	是	否	否
hireDate	入职日期	DATE	否	否	是	否	否



表 6-5 dept 表结构

字段名	字段说明	数据类型	主键	外键	非空	唯一	自增
d_no	部门编号	NUMBER (11)	是	是	是	是	是
d_name	部门名称	VARCHAR2(50)	否	否	是	否	否
d_location	部门地址	VARCHAR2(100)	否	否	否	否	否

表 6-6 employee 表中的记录

e_no	e_name	e_gender	dept_no	e_job	e_salary	hireDate
1001	SMITH	m	20	CLERK	800	2005-11-12
1002	ALLEN	f	30	SALESMAN	1600	2003-05-12
1003	WARD	f	30	SALESMAN	1250	2003-05-12
1004	JONES	m	20	MANAGER	2975	1998-05-18
1005	MARTIN	m	30	SALESMAN	1250	2001-06-12
1006	BLAKE	f	30	MANAGER	2850	1997-02-15
1007	CLARK	m	10	MANAGER	2450	2002-09-12
1008	SCOTT	m	20	ANALYST	3000	2003-05-12
1009	KING	f	10	PRESIDENT	5000	1995-01-01
1010	TURNER	f	30	SALESMAN	1500	1997-10-12
1011	ADAMS	m	20	CLERK	1100	1999-10-05
1012	JAMES	m	30	CLERK	950	2008-06-15

表 6-7 dept 表中的记录

d_no	d_name	d_location
10	ACCOUNTING	ShangHai
20	RESEARCH	BeiJing
30	SALES	ShenZhen
40	OPERATIONS	FuJian

2. 案例操作过程

步骤 01 创建数据表 employee 和 dept。

```
CREATE TABLE dept
(
  d_no      NUMBER (11) NOT NULL PRIMARY KEY,
  d_name    VARCHAR2 (50) NOT NULL,
```



```
d_location    VARCHAR2(100)
);
```

由于 employee 表 dept_no 依赖于父表 dept 的主键 d_no，因此需要先创建 dept 表，然后创建 employee 表。

```
CREATE TABLE employee
(
e_no          NUMBER(11) NOT NULL PRIMARY KEY,
e_name        VARCHAR2(30) NOT NULL,
e_gender      VARCHAR2(2) ,
dept_no       NUMBER (11) NOT NULL,
e_job         VARCHAR2(50) NOT NULL,
e_salary      NUMBER(11) NOT NULL,
hireDate      DATE NOT NULL,
CONSTRAINT dno_fk FOREIGN KEY(dept_no)
REFERENCES dept(d_no)
);
```

步骤 02 将指定记录分别插入两个表中。

向 dept 表中插入数据，SQL 语句如下：

```
INSERT INTO dept VALUES (10, 'ACCOUNTING', 'ShangHai') ;
INSERT INTO dept VALUES (20, 'RESEARCH ', 'BeiJing ') ;
INSERT INTO dept VALUES (30, 'SALES ', 'ShenZhen ') ;
INSERT INTO dept VALUES (40, 'OPERATIONS ', 'FuJian ') ;
```

向 employee 表中插入数据，SQL 语句如下：

```
INSERT INTO employee VALUES (1001, 'SMITH', 'm',20, 'CLERK',800,'12-11月-2005') ;
INSERT INTO employee VALUES (1002, 'ALLEN', 'f',30, 'SALESMAN', 1600,'12-5月-2003') ;
INSERT INTO employee VALUES (1003, 'WARD', 'f',30, 'SALESMAN', 1250,'12-5月-2003') ;
INSERT INTO employee VALUES (1004, 'JONES', 'm',20, 'MANAGER', 2975,'18-5月-1998') ;
INSERT INTO employee VALUES (1005, 'MARTIN', 'm',30, 'SALESMAN', 1250,'12-6
```



```

月-2001') ;

INSERT INTO employee VALUES (1006, 'BLAKE', 'f',30, 'MANAGER', 2850,'15-2月
-1997') ;

INSERT INTO employee VALUES (1007, 'CLARK', 'm',10, 'MANAGER', 2450,'12-9月
-2002') ;

INSERT INTO employee VALUES (1008, 'SCOTT', 'm',20, 'ANALYST', 3000,'12-5月
-2003') ;

INSERT INTO employee VALUES (1009, 'KING', 'f',10, 'PRESIDENT', 5000,'01-1月
-1995') ;

INSERT INTO employee VALUES (1010, 'TURNER', 'f',30, 'SALESMAN', 1500,'12-10
月-1997') ;

INSERT INTO employee VALUES (1011, 'ADAMS', 'm',20, 'CLERK', 1100,'05-10月
-1999') ;

INSERT INTO employee VALUES (1012, 'JAMES', 'm',30, 'CLERK', 950,'15-6月-2008');

```

步骤 03 在 employee 表中，查询所有记录的 e_no、e_name 和 e_salary 字段值。

```
SELECT e_no, e_name, e_salary FROM employee;
```

执行结果如下：

E_NO	E_NAME	E_SALARY
1001	SMITH	800
1002	ALLEN	1600
1003	WARD	1250
1004	JONES	2975
1005	MARTIN	1250
1006	BLAKE	2850
1007	CLARK	2450
1008	SCOTT	3000
1009	KING	5000
1010	TURNER	1500
1011	ADAMS	1100
1012	JAMES	950

步骤 04 在 employee 表中，查询 dept_no 等于 10 和 20 的所有记录。

```
SELECT * FROM employee WHERE dept_no IN (10, 20);
```

执行结果如下：

E_NO	E_NAME	E_GENDER	DEPT_NO	E_JOB	E_SALARY	HIREDATE
1001	SMITH	m	20	CLERK	800	12-11月-05
1004	JONES	m	20	MANAGER	2975	18-5月-98
1007	CLARK	m	10	MANAGER	2450	12-9月-02
1008	SCOTT	m	20	ANALYST	3000	12-5月-03
1009	KING	f	10	PRESIDENT	5000	01-1月-95
1011	ADAMS	m	20	CLERK	1100	05-10月-99

步骤 05 在 employee 表中，查询工资范围在 800~2500 之间的员工信息。

```
SELECT * FROM employee WHERE e_salary BETWEEN 800 AND 2500;
```

执行结果如下：

E_NO	E_NAME	E_GENDER	DEPT_NO	E_JOB	E_SALARY	HIREDATE
1001	SMITH	m	20	CLERK	800	12-11月-05
1002	ALLEN	f	30	SALESMAN	1600	12-5月-03
1003	WARD	f	30	SALESMAN	1250	12-5月-03
1005	MARTIN	m	30	SALESMAN	1250	12-6月-01
1007	CLARK	m	10	MANAGER	2450	12-9月-02
1010	TURNER	f	30	SALESMAN	1500	12-10月-97
1011	ADAMS	m	20	CLERK	1100	05-10月-99
1012	JAMES	m	30	CLERK	950	15-6月-08

步骤 06 在 employee 表中，查询部门编号为 20 的部门中的员工信息。

```
SELECT * FROM employee WHERE dept_no = 20;
```

执行结果如下：

E_NO	E_NAME	E_GENDER	DEPT_NO	E_JOB	E_SALARY	HIREDATE
1001	SMITH	m	20	CLERK	800	12-11月-05

1004	JONES	m	20	MANAGER	2975	18-5月-98
1008	SCOTT	m	20	ANALYST	3000	12-5月-03
1011	ADAMS	m	20	CLERK	1100	05-10月-99

步骤 07 在 employee 表中，查询每个部门最高工资的员工信息。

```
SELECT dept_no, MAX(e_salary) FROM employee GROUP BY dept_no;
```

执行结果如下：

DEPT_NO	MAX(E_SALARY)
10	5000
20	3000
30	2850

步骤 08 查询员工 BLAKE 所在部门和部门所在地。

```
SELECT d_no, d_location FROM dept WHERE d_no=
(SELECT dept_no FROM employee WHERE e_name='BLAKE');
```

执行结果如下：

D_NO	D_LOCATION
30	ShenZhen

步骤 09 使用连接查询，查询所有员工的部门和部门信息。

```
SELECT e_no, e_name, dept_no, d_name, d_location
FROM employee, dept WHERE dept.d_no=employee.dept_no;
```

执行结果如下：

E_NO	E_NAME	DEPT_NO	D_NAME	D_LOCATION
1001	SMITH	20	RESEARCH	BeiJing
1002	ALLEN	30	SALES	ShenZhen
1003	WARD	30	SALES	ShenZhen
1004	JONES	20	RESEARCH	BeiJing
1005	MARTIN	30	SALES	ShenZhen
1006	BLAKE	30	SALES	ShenZhen

Oracle 12c 从零开始学（视频教学版）

1007	CLARK	10	ACCOUNTING	ShangHai
1008	SCOTT	20	RESEARCH	BeiJing
1009	KING	10	ACCOUNTING	ShangHai
1010	TURNER	30	SALES	ShenZhen
1011	ADAMS	20	RESEARCH	BeiJing
1012	JAMES	30	SALES	ShenZhen

步骤 10 在 employee 表中，计算每个部门各有多少名员工。

```
SELECT dept_no, COUNT(*) FROM employee GROUP BY dept_no;
```

执行结果如下：

DEPT_NO	COUNT(*)
10	2
20	4
30	6

步骤 11 在 employee 表中，计算不同类型职工的总工资数。

```
SELECT e_job, SUM(e_salary) FROM employee GROUP BY e_job;
```

执行结果如下：

E_JOB	SUM(E_SALARY)
ANALYST	3000
CLERK	2850
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600

步骤 12 在 employee 表中，计算不同部门的平均工资。

```
SELECT dept_no, AVG(e_salary) FROM employee GROUP BY dept_no;
```

执行结果如下：

DEPT_NO	AVG(E_SALARY)
10	3725.0000


```

20      1968.7500
30      1566.6667

```

步骤 13 在 employee 表中，查询工资低于 1500 的员工信息。

```

SELECT e_no,e_name, e_gender ,dept_no ,e_job,e_salary FROM employee WHERE
e_salary < 1500;

```

执行结果如下：

E_NO	E_NAME	E_GENDER	DEPT_NO	E_JOB	E_SALARY
1001	SMITH	m	20	CLERK	800
1003	WARD	f	30	SALESMAN	1250
1005	MARTIN	m	30	SALESMAN	1250
1011	ADAMS	m	20	CLERK	1100
1012	JAMES	m	30	CLERK	950

步骤 14 在 employee 表中，将查询记录先按部门编号由高到低排列，再按员工工资由高到低排列。

```

SELECT e_name,dept_no, e_salary
FROM employee ORDER BY dept_no DESC, e_salary DESC;

```

执行结果如下：

E_NAME	DEPT_NO	E_SALARY
BLAKE	30	2850
ALLEN	30	1600
TURNER	30	1500
WARD	30	1250
MARTIN	30	1250
JAMES	30	950
SCOTT	20	3000
JONES	20	2975
ADAMS	20	1100
SMITH	20	800
KING	10	5000

CLARK	10	2450
-------	----	------

步骤 15 在 employee 表中，查询员工姓名以字母“A”开头的员工的信息。

```
SELECT e_no , e_name FROM employee WHERE REGEXP_LIKE(e_name , '^A');
```

执行过程如下：

E_NO	E_NAME
1002	ALLEN
1011	ADAMS

6.10 疑难解惑

疑问 1：在 WHERE 子句中必须使用圆括号吗？

任何时候使用具有 AND 和 OR 操作符的 WHERE 子句，都应该使用圆括号明确操作顺序。如果条件较多，即使能确定计算次序，默认的计算次序也可能会使 SQL 语句不易理解，因此使用圆括号明确操作符的次序，是一个好的习惯。

疑问 2：为什么使用通配符格式正确，却没有查找出符合条件的记录？

Oracle 在存储字符串数据时，可能会不小心把两端带有空格的字符串保存到记录中，而在查看表中记录时，Oracle 不能明确地显示空格，数据库操作者不能直观地确定字符串两端是否有空格。例如，使用 LIKE '%e' 匹配以字母 e 结尾的水果的名称，如果字母 e 后面多了一个空格，则 LIKE 语句不能将该记录查找出来。解决的方法是使用 TRIM 函数，将字符串两端的空格删除之后再行匹配。

6.11 经典习题

在已经创建的 employee 表中进行如下操作：

- (1) 计算所有女员工 (e-gendev= 'F') 的年龄。
- (2) 使用 ROWNUM 查询从第 3 条记录开始的记录。
- (3) 查询销售人员 (SALESMAN) 的最低工资。

- (4) 查询员工姓名以字母“N”或者“S”结尾的记录。
- (5) 查询在 BeiJing 工作的员工的姓名和职务。
- (6) 使用左连接方式查询 employee 和 dept 表。
- (7) 查询所有 2001~2005 年入职的员工的信息，查询部门编号为 20 和 30 的员工信息并使用 UNION 合并两个查询结果。
- (8) 使用 LIKE 查询员工姓名中包含字母“a”的记录。
- (9) 使用 REGEXP_LIKE 函数查询员工姓名中包含 T、C 或者 M 三个字母中任意一个的记录。

第 7 章

◀ 插入、更新与删除数据 ▶

存储在系统中的数据是数据库管理系统（DBMS）的核心，数据库被设计用来管理数据的存储、访问和维护数据的完整性。Oracle 中提供了功能丰富的数据库管理语句，包括有效的向数据库中插入数据的 INSERT 语句，更新数据的 UPDATE 语句，以及当数据不再使用时删除数据的 DELETE 语句。本章将详细介绍在 Oracle 中如何使用这些语句操作数据。

- 掌握如何向表中插入数据
- 掌握更新数据的方法
- 熟悉如何删除数据
- 掌握综合案例中数据的基本操作方法和技巧

7.1 插入数据

在使用数据库之前，数据库中必须要有数据，Oracle 中使用 INSERT 语句向数据表中插入新的数据记录。可以插入的方式有：插入完整的记录、插入记录的一部分、插入多条记录、插入另一个查询的结果，下面将分别介绍这些内容。

7.1.1 为表的所有字段插入数据

使用基本的 INSERT 语句插入数据要求指定表名称和插入到新记录中的值。基本语法格式为：

```
INSERT INTO table_name (column_list) VALUES (value_list);
```

table_name 指定要插入数据的表名，column_list 指定要插入数据的那些列，value_list 指定每个列应对应插入的数据。注意，使用该语句时字段列和数据值的数量必须相同。

本章将使用样例表 person，创建语句如下：

```
CREATE TABLE person
(
    id    NUMBER(9) GENERATED BY DEFAULT AS IDENTITY,
```

```
name VARCHAR2(40) NOT NULL,
age   NUMBER(9)   NOT NULL ,
info  VARCHAR2(50) NULL,
PRIMARY KEY (id)
);
```

向表中所有字段插入值的方法有两种：一种是指定所有字段名，另一种是完全不指定字段名。

【例 7.1】在 person 表中，插入一条新记录，id 值为 1，name 值为 Green，age 值为 21，info 值为 Lawyer，SQL 语句如下：

执行插入操作之前，使用 SELECT 语句查看表中的数据：

```
SQL> SELECT * FROM person;
```

未选择任何行

结果显示当前表为空，没有数据，接下来执行插入操作：

```
SQL> INSERT INTO person (id ,name, age , info) VALUES (1, 'Green', 21, 'Lawyer');
```

语句执行完毕，查看执行结果：

```
SQL> SELECT * FROM person;
```

```
ID NAME AGE INFO
```

```
-----
1 Green 21 Lawyer
```

可以看到插入记录成功。在插入数据时，指定了 person 表的所有字段，因此将为每一个字段插入新的值。

INSERT 语句后面的列名称顺序可以不是 person 表定义时的顺序。即插入数据时，不需要按照表定义的顺序插入，只要保证值的顺序与列字段的顺序相同就可以，如【例 7.2】所示。

【例 7.2】在 person 表中，插入一条新记录，id 值为 2，name 值为 Suse，age 值为 22，info 值为 dancer，SQL 语句如下：

```
SQL> INSERT INTO person (age ,name, id , info) VALUES (22, 'Suse', 2, 'dancer');
```

语句执行完毕，查看执行结果：

```
SQL> SELECT * FROM person;
```

```
ID NAME AGE INFO
```

```
-----
```


1	Green	21	Lawyer
2	Suse	22	dancer

由结果可以看到，INSERT 语句成功插入了一条记录。

使用 INSERT 插入数据时，允许列名称列表 column_list 为空，此时，值列表中需要为表的每一个字段指定值，并且值的顺序必须和数据表中字段定义时的顺序相同，如【例 7.3】所示。

【例 7.3】在 person 表中，插入一条新记录，id 值为 3，name 值为 Mary，age 值为 24，info 值为 Musician，SQL 语句如下：

```
SQL> INSERT INTO person
      VALUES (3, 'Mary', 24, 'Musician');
```

语句执行完毕，查看执行结果：

```
SQL> SELECT * FROM person;

ID NAME AGE INFO
---
1  Green  21  Lawyer
2  Suse   22  dancer
3  Mary   24  Musician
```

可以看到插入记录成功。数据库中增加了一条 id 为 3 的记录，其他字段值为指定的插入值。本例的 INSERT 语句中没有指定插入列表，只有一个值列表。在这种情况下，值列表为每一个字段列指定插入值，并且这些值的顺序必须和 person 表中字段定义的顺序相同。

提示

虽然使用 INSERT 插入数据时可以忽略插入数据的列名称，如果不包含列名称，那么 VALUES 关键字后面的值不仅要求完整且顺序必须和表定义时列的顺序相同。如果表的结构被修改，对列进行增加、删除或者位置改变操作，这些操作将使得用这种方式插入数据时的顺序也同时改变。如果指定列名称，则不会受到表结构改变的影响。

7.1.2 为表的指定字段插入数据

为表的指定字段插入数据，就是在 INSERT 语句中只向部分字段中插入值，而其他字段的值为表定义时的默认值。

【例 7.4】在 person 表中，插入一条新记录，name 值为 Willam，age 值为 20，info 值为 sports man，SQL 语句如下：

```
SQL> INSERT INTO person (name, age, info) VALUES ('Willam', 20, 'sports man');
```


提示信息表示插入一条记录成功。使用 SELECT 查询表中的记录，查询结果如下：

```
SQL> SELECT * FROM person;
```

ID	NAME	AGE	INFO
1	Green	21	Lawyer
2	Suse	22	dancer
3	Mary	24	Musician
4	Willam	20	sports man

可以看到插入记录成功。在这里的 id 字段，如查询结果显示，该字段自动添加了一个整数 4。在这里 id 字段为表的主键，不能为空，系统会自动为该字段插入自增的序列值。在插入记录时，如果某些字段没有指定插入值，Oracle 将插入该字段定义时的默认值。下面例子说明在没有指定字段值时，插入默认值。

【例 7.5】在 person 表中，插入一条新记录，name 值为 laura，age 值为 25，SQL 语句如下：

```
SQL> INSERT INTO person (name, age ) VALUES ('Laura', 25);
```

语句执行完毕，查看执行结果：

```
SQL> SELECT * FROM person;
```

ID	NAME	AGE	INFO
1	Green	21	Lawyer
2	Suse	22	dancer
3	Mary	24	Musician
4	Willam	20	sports man
5	Laura	25	

可以看到，在本例插入语句中，没有指定 info 字段值，查询结果显示，info 字段在定义时默认为空，因此系统自动为该字段插入空值。



要保证每个插入值的类型和对应列的数据类型匹配，如果类型不同，将无法插入，并且 Oracle 会产生错误。

7.1.3 同时插入多条记录

使用多个 INSERT 语句可以向数据表中插入多条记录。

【例 7.6】在 person 表中，在 name、age 和 info 字段指定插入值，插入 3 条新记录，SQL 语句如下：

```
INSERT INTO person(name, age, info)
VALUES ('Evans',27, 'secretary') ;
INSERT INTO person(name, age, info)
VALUES('Dale',22, 'cook') ;
INSERT INTO person(name, age, info)
VALUES('Edison',28, 'singer');
```

语句执行完毕，查看执行结果：

```
SQL> SELECT * FROM person;
```

ID	NAME	AGE	INFO
1	Green	21	Lawyer
2	Suse	22	dancer
3	Mary	24	Musician
4	Willam	20	sports man
5	Laura	25	
6	Evans	27	secretary
7	Dale	22	cook
8	Edison	28	singer

由结果可以看到，INSERT 语句执行后，person 表中添加了 3 条记录，其 name、age 和 info 字段分别为指定的值，id 字段为 Oracle 添加的默认的自增值。

如果想使用 INSERT 同时插入多条记录，需要配合 SELECT 同时操作才行。

【例 7.7】在 person 表中，不指定插入列表，同时插入 2 条新记录，SQL 语句如下：

```
INSERT INTO person (id,name, age, info)
SELECT 9,'Harry',21, 'magician' from dual
Union all
SELECT 10,'Harriet',19, 'pianist' from dual;
```

语句执行完毕，查看执行结果：

```
SQL> SELECT * FROM person;
```

ID	NAME	AGE	INFO
1	Green	21	Lawyer
2	Suse	22	dancer
3	Mary	24	Musician
4	Willam	20	sports man
5	Laura	25	
6	Evans	27	secretary
7	Dale	22	cook
8	Edison	28	singer
9	Harry	21	magician
10	Harriet	19	pianist

由结果可以看到, INSERT 语句执行后, person 表中添加了 2 条记录。

提示

一个同时插入多行记录的 INSERT 语句可以等同于多个单行插入的 INSERT 语句,但是多行的 INSERT 语句在处理过程中,效率更高。因为 Oracle 执行单条 INSERT 语句插入多行数据,比使用多条 INSERT 语句快。所以在插入多条记录时,最好选择使用单条 INSERT 语句的方式插入。

7.1.4 将查询结果插入到表中

INSERT 语句用来给数据表插入记录时,指定插入记录的列值。INSERT 还可以将 SELECT 语句查询的结果插入到表中,如果要从另外一个表中合并个人信息到 person 表,不需要把每一条记录的值一个一个输入,只需要使用一条 INSERT 语句和一条 SELECT 语句组成的组合语句,即可快速地从 一个或多个表中向一个表中插入多个行。基本语法格式如下:

```
INSERT INTO table_name1 (column_list1)
SELECT (column_list2) FROM table_name2 WHERE (condition)
```

table_name1 指定待插入数据的表; column_list1 指定待插入表中要插入数据的那些列; table_name2 指定插入数据是从哪个表中查询出来的; column_list2 指定数据来源表的查询列,该列表必须和 column_list1 列表中的字段个数相同,数据类型相同; condition 指定 SELECT 语句的查询条件。

【例 7.8】从 person_old 表中查询所有的记录,并将其插入到 person 表中。

首先,创建一个名为 person_old 的数据表,其表结构与 person 结构相同,SQL 语句如下:

```
CREATE TABLE person_old
```

```
(
```



```

id    NUMBER(9) GENERATED BY DEFAULT AS IDENTITY,
name  VARCHAR2(40) NOT NULL,
age    NUMBER(9) NOT NULL ,
info  VARCHAR2(50) NULL,
PRIMARY KEY (id)
);

```

向 person_old 表中添加两条记录:

```

SQL> INSERT INTO person_old VALUES (11,'Harry',20, 'student');
SQL>INSERT INTO person_old VALUES (12,' Beckham ',31, 'police');

```

查询 person_old 表中的记录, 结果如下:

```
SQL> SELECT * FROM person_old;
```

ID	NAME	AGE	INFO
11	Harry	20	student
12	Beckham	31	police

可以看到, 插入记录成功, person_old 表中现在有两条记录。接下来将 person_old 表中所有的记录插入 person 表中, SQL 语句如下:

```

INSERT INTO person(id, name, age, info)
SELECT id, name, age, info FROM person_old;

```

语句执行完毕, 查看执行结果:

```
SQL> SELECT * FROM person;
```

ID	NAME	AGE	INFO
1	Green	21	Lawyer
2	Suse	22	dancer
3	Mary	24	Musician
4	Willam	20	sports man
5	Laura	25	
6	Evans	27	secretary


```

7   Dale      22   cook
8   Edison    28   singer
9   Harry     21   magician
10  Harriet   19   pianist
11  Harry     20   student
12  Beckham  31   police

```

由结果可以看到, INSERT 语句执行后, person 表中多了两条记录, 这两条记录和 person_old 表中的记录完全相同, 数据转移成功。这里的 id 字段为自增的主键, 在插入的时候要保证该字段值的唯一性, 如果不能确定, 可以插入的时候忽略该字段, 只插入其他字段的值。

提示

这个例子中使用的 person_old 表和 person 表的定义相同, 事实上, Oracle 不关心 SELECT 返回的列名, 它根据列的位置进行插入, SELECT 的第 1 列对应待插入表的第 1 列, 第 2 列对应待插入表的第 2 列, 等等。即使不同结果的表之间也可以方便地转移数据。

7.2 更新数据

表中有数据之后, 接下来可以对数据进行更新操作, Oracle 中使用 UPDATE 语句更新表中的记录, 可以更新特定的行或者同时更新所有的行。基本语法结构如下:

```

UPDATE table_name
SET column_name1 = value1, column_name2=value2,..., column_namen=valuen
WHERE (condition);

```

column_name1, column_name2,..., column_namen 为指定更新的字段的名称; value1, value2,..., valuen 为相对应的指定字段的更新值; condition 指定更新的记录需要满足的条件。更新多个列时, 每个“列-值”对之间用逗号隔开, 最后一列之后不需要逗号。

【例 7.9】在 person 表中, 更新 id 值为 11 的记录, 将 age 字段值改为 15, 将 name 字段值改为 LiMing, SQL 语句如下:

```
UPDATE person SET age = 15, name='LiMing' WHERE id = 11;
```

更新操作执行前, 可以使用 SELECT 语句查看当前的数据:

```
SQL> SELECT * FROM person WHERE id=11;
```

```

ID  NAME  AGE  INFO
----
11  Harry  20   student

```




由结果可以看到更新之前, id 等于 11 的记录的 name 字段值为 Harry, age 字段值为 20。下面使用 UPDATE 语句更新数据, 语句执行结果如下:

```
SQL> UPDATE person SET age = 15, name='LiMing' WHERE id = 11;
```

语句执行完毕, 查看执行结果:

```
SQL> SELECT * FROM person WHERE id=11;
```

ID	NAME	AGE	INFO
11	LiMing	15	student

由结果可以看到, id 等于 11 的记录中的 name 和 age 字段的值已经成功被修改为指定值。



保证 UPDATE 以 WHERE 子句结束, 通过 WHERE 子句指定被更新的记录所需要满足的条件, 如果忽略 WHERE 子句, Oracle 将更新表中所有的行。

【例 7.10】在 person 表中, 更新 age 值为 19~22 的记录, 将 info 字段值都改为 student, SQL 语句如下:

```
UPDATE person SET info='student' WHERE age BETWEEN 19 AND 22;
```

更新操作执行前, 可以使用 SELECT 语句查看当前的数据:

```
SQL> SELECT * FROM person WHERE age BETWEEN 19 AND 22;
```

ID	NAME	AGE	INFO
1	Green	21	Lawyer
2	Suse	22	dancer
4	Willam	20	sports man
7	Dale	22	cook
9	Harry	21	magician
10	Harriet	19	pianist

可以看到, 这些 age 字段值在 19~22 之间的记录的 info 字段值各不相同。下面使用 UPDATE 语句更新数据, 语句执行结果如下:

```
SQL> UPDATE person SET info='student' WHERE age BETWEEN 19 AND 22;
```

语句执行完毕, 查看执行结果:



```
SQL> SELECT * FROM person WHERE age BETWEEN 19 AND 22;
```

ID	NAME	AGE	INFO
1	Green	21	student
2	Suse	22	student
4	Willam	20	student
7	Dale	22	student
9	Harry	21	student
10	Harriet	19	student

由结果可以看到，UPDATE 执行后，成功将表中符合条件的 6 条记录的 info 字段值都改为 student。

7.3 删除数据

从数据表中删除数据使用 DELETE 语句，DELETE 语句允许 WHERE 子句指定删除条件。DELETE 语句基本语法格式如下：

```
DELETE FROM table_name [WHERE <condition>];
```

table_name 指定要执行删除操作的表；“[WHERE <condition>]”为可选参数，指定删除条件，如果没有 WHERE 子句，DELETE 语句将删除表中的所有记录。

【例 7.11】在 person 表中，删除 id 等于 11 的记录，SQL 语句如下：

```
SQL> DELETE FROM person WHERE id=11;
```

执行删除操作前，使用 SELECT 语句查看当前 id=11 的记录：

```
SQL> SELECT * FROM person WHERE id=11;
```

ID	NAME	AGE	INFO
11	LiMing	15	student

可以看到，现在表中有 id=11 的记录。下面使用 DELETE 语句删除该记录，语句执行结果如下：

```
SQL> DELETE FROM person WHERE id = 11;
```




1 行已删除。

语句执行完毕，查看执行结果：

```
SQL> SELECT * FROM person WHERE id=11;
```

未选择任何行

查询结果为空，说明删除操作成功。

【例 7.12】在 person 表中，使用 DELETE 语句同时删除多条记录，在前面 UPDATE 语句中将 age 字段值在 19~22 之间的记录的 info 字段值修改为 student，在这里删除这些记录，SQL 语句如下：

```
DELETE FROM person WHERE age BETWEEN 19 AND 22;
```

执行删除操作前，使用 SELECT 语句查看当前的数据：

```
SQL> SELECT * FROM person WHERE age BETWEEN 19 AND 22;
```

ID	NAME	AGE	INFO
1	Green	21	student
2	Suse	22	student
4	Willam	20	student
7	Dale	22	student
9	Harry	21	student
10	Harriet	19	student

可以看到，这些 age 字段值在 19~22 之间的记录存在表中。下面使用 DELETE 删除这些记录：

```
SQL> DELETE FROM person WHERE age BETWEEN 19 AND 22;
```

语句执行完毕，查看执行结果：

```
SQL> SELECT * FROM person WHERE age BETWEEN 19 AND 22;
```

未选择任何行

查询结果为空，删除多条记录成功。

【例 7.13】删除 person 表中所有记录，SQL 语句如下：

```
DELETE FROM person;
```

执行删除操作前，使用 SELECT 语句查看当前的数据：



```
SQL> SELECT * FROM person;
```

ID	NAME	AGE	INFO
3	Mary	24	Musician
5	Laura	25	
6	Evans	27	secretary
8	Eclison	28	singer
12	Beckham	31	police

结果显示 person 表中还有 5 条记录，执行 DELETE 语句删除这 5 条记录：

```
SQL> DELETE FROM person;
```

语句执行完毕，查看执行结果：

```
SQL> SELECT * FROM person;
```

未选择任何行

查询结果为空，删除表中所有记录成功，现在 person 表中已经没有任何数据记录。

提示

如果想删除表中的所有记录，还可以使用 TRUNCATE TABLE 语句，TRUNCATE 将直接删除原来的表并重新创建一个表，其语法结构为 TRUNCATE TABLE table_name。TRUNCATE 直接删除表而不是删除记录，因此执行速度比 DELETE 快。

7.4 综合案例——记录的插入、更新和删除

本章重点介绍了数据表中数据的插入、更新和删除操作。Oracle 中可以灵活地对数据进行插入与更新，Oracle 中对数据的操作没有任何提示，因此在更新和删除数据时，一定要谨慎小心，查询条件一定要准确，避免造成数据的丢失。本章的综合案例包含了对数据表中数据的基本操作，包括记录的插入、更新和删除。

1. 案例目的

创建表 books，对数据表进行插入、更新和删除操作，掌握表数据基本操作。books 表结构以及表中的记录，如表 7-1 和表 7-2 所示。



表 7-1 books 表结构

字段名	字段说明	数据类型	主键	外键	非空	唯一	自增
id	书编号	NUMBER(11)	是	否	是	是	否
name	书名	VARCHAR2(50)	否	否	是	否	否
authors	作者	VARCHAR2(100)	否	否	是	否	否
price	价格	NUMBER(11)	否	否	是	否	否
pubdate	出版日期	DATE	否	否	是	否	否
note	说明	VARCHAR2(100)	否	否	否	否	否
num	库存	NUMBER(11)	否	否	是	否	否

表 7-2 books 表中的记录

b_id	b_name	Authors	price	pubdate	note	num
1	Tale of AAA	Dickes	23	1995-10-01	novel	11
2	EmmaT	Jane lura	35	1993-10-01	joke	22
3	Story of Jane	Jane Tim	40	2001-10-01	novel	0
4	Lovey Day	George Byron	20	2005-10-01	novel	30
5	Old Land	Honore Blade	30	2010-10-01	law	0
6	The Battle	Upton Sara	33	1999-10-01	medicine	40
7	Rose Hood	Richard Kale	28	2008-10-01	cartoon	28

2. 案例操作过程

步骤 01 创建数据表 books，并按表 7-1 结构定义各个字段。

```
CREATE TABLE books
(
  id      NUMBER(11) GENERATED BY DEFAULT AS IDENTITY,
  name    VARCHAR2(50) NOT NULL,
  authors VARCHAR2(100) NOT NULL,
  price   NUMBER(11) NOT NULL,
  pubdate DATE NOT NULL,
  note    VARCHAR2(100) NULL,
  num     NUMBER(11) DEFAULT 0
);
```

步骤 02 将表 7-2 中的记录插入 books 表中，分别使用不同的方法插入记录，执行过程如下。



表创建好之后，使用 SELECT 语句查看表中的数据，结果如下：

```
SQL> SELECT * FROM books;
```

未选择任何行

可以看到，当前表中为空，没有任何数据，下面向表中插入记录。

(1) 指定所有字段名称插入记录，SQL 语句如下：

```
SQL> INSERT INTO books (id, name, authors, price, pubdate,note,num)
VALUES (1, 'Tale of AAA', 'Dickes', 23, '01-10月-1995', 'novel',11);
```

语句执行成功，插入了一条记录。

(2) 不指定字段名称插入记录，SQL 语句如下：

```
SQL> INSERT INTO books
VALUES (2,'EmmaT','Jane lura',35,'01-10月-1993', 'joke',22);
```

语句执行成功，插入了一条记录。

使用 SELECT 语句查看当前表中的数据：

```
SQL> SELECT * FROM books;
```

ID	NAME	AUTHORS	PRICE	PUBDATE	NOTE	NUM
1	Tale of AAA	Dickes	23	01-10月-95	novel	11
2	EmmaT	Jane lura	35	01-10月-93	joke	22

可以看到，两条语句分别成功插入了两条记录。

(3) 同时插入多条记录。

使用 INSERT 语句将剩下的多条记录插入表中，SQL 语句如下：

```
SQL> INSERT INTO books
SELECT 3, 'Story of Jane', 'Jane Tim', 40, '01-10月-2001', 'novel', 0 from dual
Union all
SELECT 4, 'Lovey Day', 'George Byron', 20, '01-10月-2005', 'novel', 30 from dual
Union all
SELECT 5, 'Old Land', 'Honore Blade', 30, '01-10月-2010', 'law',0 from dual
Union all
SELECT 6, 'The Battle', 'Upton Sara',33,'01-10月-1999', 'medicine',40 from dual
```




```
Union all
```

```
SELECT 7, 'Rose Hood', 'Richard Kale', 28, '01-10月-2008', 'cartoon', 28 from dual;
```

5 行已插入。

由结果可以看到，语句执行成功，总共插入了 5 条记录，使用 SELECT 语句查看表中所有的记录：

```
SQL> SELECT * FROM books;
```

ID	NAME	AUTHORS	PRICE	PUBDATE	NOTE	NUM
1	Tale of AAA	Dickes	23	01-10月- 95	novel	11
2	EmmaT	Jane lura	35	01-10月- 3	joke	22
3	Story of Jane	Jane Tim	40	01-10月- 01	novel	0
4	Lovey Day	George Byron	20	01-10月- 05	novel	30
5	Old Land	Honore Blade	30	01-10月- 10	law	0
6	The Battle	Upton Sara	33	01-10月- 99	medicine	40
7	Rose Hood	Richard Kale	28	01-10月- 08	cartoon	28

由结果可以看到，所有记录成功插入表中。

步骤 03 将小说类型（novel）的书的价格都增加 5。

执行该操作的 SQL 语句为：

```
UPDATE books SET price = price + 5 WHERE note = 'novel';
```

执行前先使用 SELECT 语句查看当前记录：

```
SQL> SELECT id, name, price, note FROM books WHERE note = 'novel';
```

ID	NAME	PRCE	NOTE
1	Tale of AAA	23	novel
3	Story of Jane	40	novel
4	Lovey Day	20	novel

使用 UPDATE 语句执行更新操作：

```
SQL> UPDATE books SET price = price + 5 WHERE note = 'novel';
```

由结果可以看到，该语句对 3 条记录进行了更新，使用 SELECT 语句查看更新结果：



```
SQL> SELECT id, name, price, note FROM books WHERE note = 'novel';
```

ID	NAME	PRICE	NOTE
1	Tale of AAA	28	novel
3	Story of Jane	45	novel
4	Lovey Day	25	novel

对比可知, price 的值都在原来的价格之上增加了 5。

步骤 04 将名称为 EmmaT 的书的价格改为 40, 并将说明改为 drama。

修改语句为:

```
UPDATE books SET price=40,note= 'drama 'WHERE name= 'EmmaT ';
```

执行修改前, 使用 SELECT 语句查看当前记录:

```
SQL> SELECT name, price, note FROM books WHERE name='EmmaT';
```

NAME	PRICE	NOTE
EmmaT	35	joke

下面执行修改操作:

```
SQL> UPDATE books SET price=40,note='drama' WHERE name='EmmaT';
```

结果显示修改了一条记录, 使用 SELECT 查看执行结果:

```
SQL> SELECT name, price, note FROM books WHERE name='EmmaT';
```

NAME	PRICE	NOTE
EmmaT	40	drama

可以看到, price 和 note 字段的值已经改变, 修改操作成功。

步骤 05 删除库存为 0 的记录。

删除库存为 0 的语句为:

```
DELETE FROM books WHERE num=0;
```




删除之前使用 SELECT 语句查看当前记录：

```
SQL> SELECT * FROM books WHERE num=0;
```

ID	NAME	AUTHORS	PRICE	PUBDATE	NOTE	NUM
3	Story of Jane	Jane Tim	40	01-10月- 01	novel	0
5	Old Land	Honore Blade	30	01-10月- 10	law	0

可以看到，当前有两条记录的 num 值为 0，下面使用 DELETE 语句删除这两条记录，SQL 语句如下：

```
SQL> DELETE FROM books WHERE num=0;
```

语句执行成功，查看操作结果：

```
SQL> SELECT * FROM books WHERE num=0;
```

未选择任何行

可以看到，查询结果为空，表中已经没有库存量为 0 的记录。

7.5 疑难解惑

疑问 1：插入记录时可以不指定字段名称吗？

不管使用哪种 INSERT 语法，都必须给出 VALUES 的正确数目。如果不提供字段名，则必须给每个字段提供一个值，否则将产生一条错误消息。如果要在 INSERT 操作中省略某些字段，这些字段需要满足一定条件：该列定义为允许空值；或者表定义时给出默认值，如果不给出值，将使用默认值。

疑问 2：更新或者删除表时必须指定 WHERE 子句吗？

在前面章节中可以看到，所有的 UPDATE 和 DELETE 语句全都在 WHERE 子句中指定了条件。如果省略 WHERE 子句，则 UPDATE 或 DELETE 将被应用到表中所有的行。因此，除非确实打算更新或者删除所有记录，否则要谨慎使用不带 WHERE 子句的 UPDATE 或 DELETE 语句。建议在对表进行更新和删除操作之前，使用 SELECT 语句确认需要删除的记录，以免造成无法挽回的后果。



7.6 经典习题

创建数据表 pet，并对表进行插入、更新与删除操作，pet 表结构如表 7-3 所示。

- (1) 首先创建数据表 pet，使用不同的方法将表 7-4 中的记录插入到 pet 表中。
- (2) 使用 UPDATE 语句将名称为 Fang 的狗的主人改为 Kevin。
- (3) 将没有主人的宠物的 owner 字段值都改为 Duck。
- (4) 删除已经死亡的宠物记录。
- (5) 删除所有表中的记录。

表 7-3 pet 表结构

字段名	字段说明	数据类型	主键	外键	非空	唯一	自增
name	宠物名称	VARCHAR2(20)	否	否	是	否	否
owner	宠物主人	VARCHAR2(20)	否	否	否	否	否
species	种类	VARCHAR2(20)	否	否	是	否	否
sex	性别	VARCHAR2(1)	否	否	是	否	否
birth	出生日期	DATE	否	否	是	否	否
death	死亡日期	DATE	否	否	否	否	否

表 7-4 pet 表中记录

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	2003-10 月 12	2010-8 月 12
Claws	Gwen	cat	m	2004-8 月 10	NULL
Buffy	NULL	dog	f	2009-8 月 11	NULL
Fang	Benny	dog	m	2000-5 月 15	NULL
Bowser	Diane	dog	m	2003-4 月 16	2009-11 月 12
Chirpy	NULL	bird	f	2008-5 月 19	NULL



第 8 章

◀ 视 图 ▶

数据库中的视图是一个虚拟表。同真实的表一样, 视图包含一系列带有名称的行和列数据。行和列数据来自自定义视图查询所引用的表, 并且在引用视图时动态生成。本章将通过一些实例来介绍视图的含义、视图的作用、创建视图、查看视图、修改视图、更新视图和删除视图等 Oracle 的数据库知识。

- 了解视图的含义和作用
- 掌握创建视图的方法
- 熟悉如何查看视图
- 掌握修改视图的方法
- 掌握更新视图的方法
- 掌握删除视图的方法
- 掌握综合案例中视图应用的方法和技巧

8.1 视图概述

视图是从一个或者多个表中导出的, 视图的行为与表非常相似, 但视图是一个虚拟表。在视图中用户可以使用 `SELECT` 语句查询数据, 以及使用 `INSERT`、`UPDATE` 和 `DELETE` 修改记录。视图可以使用户操作方便, 而且可以保障数据库系统的安全。

8.1.1 视图的含义

视图是一个虚拟表, 是从数据库的一个或多个表中导出来的。视图还可以从已经存在的视图的基础上定义。

视图一经定义便存储在数据库中, 与其相对应的数据并没有像表那样在数据库中再存储一份, 通过视图看到的数据只是存放在基本表中的数据。对视图的操作与对表的操作一样, 可以对其进行查询、修改和删除。当对通过视图看到的数据进行修改时, 相应的基本表的数据也要发生变化; 同时, 若基本表的数据发生变化, 则这种变化也可以自动地反映到视图中。

下面有个 `student` 表和 `stu_info` 表, 在 `student` 表中包含了学生的 `id` 号和姓名, `stu_info` 包



含了学生的 id 号、班级和家庭住址，而现在公布分班信息，只需要 id 号、姓名和班级，这该如何解决？通过学习后面的内容就可以找到完美的解决方案。

表设计如下：

```
CREATE TABLE student
(
    s_id NUMBER(9),
    name VARCHAR2(40)
);

CREATE TABLE stu_info
(
    s_id NUMBER(9),
    glass VARCHAR2(40),
    addr VARCHAR2(90)
);
```

通过 DESC 命令可以查看表的设计，可以获得字段、字段的定义、是否为主键、是否为空、默认值和扩展信息。

视图提供了一个很好的解决方法，创建一个视图，这些信息来自表的部分信息，其他的信息不取，这样既能满足要求也不破坏表原来的结构。

8.1.2 视图的作用

与直接从数据表中读取相比，视图有以下优点。

1. 简单化

看到的就是需要的。视图不仅可以简化用户对数据的理解，也可以简化他们的操作。那些被经常使用的查询可以被定义为视图，从而使得用户不必为以后的操作每次指定全部的条件。

2. 安全性

通过视图用户只能查询和修改他们所能见到的数据，数据库中的其他数据则既看不见也取不到。数据库授权命令可以使每个用户对数据库的检索限制到特定的数据库对象上，但不能授权到数据库特定行和特定列上。通过视图，用户可以被限制在数据的不同子集上。

- (1) 使用权限可被限制在基表的行的子集上。
- (2) 使用权限可被限制在基表的列的子集上。
- (3) 使用权限可被限制在基表的行和列的子集上。
- (4) 使用权限可被限制在多个基表的连接所限定的行上。



(5) 使用权限可被限制在基表中的数据的统计汇总上。

(6) 使用权限可被限制在另一视图的一个子集上，或是一些视图和基表合并后的子集上。

3. 逻辑数据独立性

视图可帮助用户屏蔽真实表结构变化带来的影响。

8.2 创建视图

视图中包含了 SELECT 查询的结果，因此视图的创建基于 SELECT 语句和已存在的数据表，视图可以建立在一张表上，也可以建立在多张表上。本节主要介绍创建视图的方法。

8.2.1 创建视图的语法形式

创建视图使用 CREATE VIEW 语句，基本语法格式如下：

```
CREATE [OR REPLACE] [[NO]FORCE] VIEW
    [schema.] view
    [(alias,...)inline_constraint(s)]
    [out_of_line_constraint (s)]
AS subquery
[
    WITH{READ ONLY CHECK OPTION[CONSTRAINT constraint]}
];
```

其中，CREATE 表示创建新的视图；REPLACE 表示替换已经创建的视图；[NO]FORCE 表示是否强制创建视图；[schema.] view 表示视图所属方案的名称和视图本身的名称；[(alias,...)inline_constraint(s)]表示视图字段的别名和内联的名称；[out_of_line_constraint (s)]表示约束，是与 inline_constraint(s)相反的声明方式；WITH READ ONLY 表示视图为只读；WITH CHECK OPTION 表示一旦使用该限制，当对视图增加或修改数据时必须满足子查询的条件。

8.2.2 在单表上创建视图

Oracle 可以在单个数据表上创建视图。

【例 8.1】在 t 表格上创建一个名为 view_t 的视图，代码如下：

首先创建基本表并插入数据，语句如下：

```
CREATE TABLE t (quantity NUMBER(9), price NUMBER(9));
INSERT INTO t VALUES(3, 50);
```



创建视图语句为：

```
CREATE VIEW view_t AS SELECT quantity, price FROM t;
```

语句执行如下：

```
SQL> SELECT * FROM view_t;
```

QUANTITY	PRICE
3	50

默认情况下创建的视图和基本表的字段是一样的，也可以通过指定视图字段的名称来创建视图。

【例 8.2】在 t 表格上创建一个名为 view_t2 的视图，代码如下：

```
SQL> CREATE VIEW view_t2(qty, price ) AS SELECT quantity, price FROM t;
```

语句执行成功，查看 view_t2 视图中的数据：

```
SQL> SELECT * FROM view_t2;
```

QTY	PRICE
3	50

可以看到，view_t2 和 view_t 两个视图中字段名称不同，但数据却是相同的。因此，在使用视图的时候，可能用户根本就不需要了解基本表的结构，更接触不到实际表中的数据，从而保证了数据库的安全。

8.2.3 在多表上创建视图

Oracle 也可以在两个或者两个以上的表上创建视图，可以使用 CREATE VIEW 语句实现。

【例 8.3】在表 student 和表 stu_info 上创建视图 stu_glass，代码如下：

首先向两个表中插入数据，输入语句如下：

```
SQL> INSERT INTO student VALUES(1,'wanglin1');
```

```
SQL> INSERT INTO student VALUES(2,'gaoli');
```

```
SQL> INSERT INTO student VALUES(3,'zhanghai');
```

```
SQL> INSERT INTO stu_info VALUES(1, 'wuban', 'henan');
```




```
SQL> INSERT INTO stu_info VALUES (2,'liuban','hebei') ;
SQL> INSERT INTO stu_info VALUES (3,'qiban','shandong');
```

创建视图 `stu_glass`，SQL 语句如下：

```
CREATE VIEW stu_glass (id,name, glass) AS SELECT
student.s_id,student.name ,stu_info.glass
FROM student ,stu_info WHERE student.s_id=stu_info.s_id;
```

查询视图 `stu_glass`，SQL 语句如下：

```
SQL> SELECT * FROM stu_glass;
```

```
ID NAME GLASS
```

```
-----
```

```
1 wanglin1 wuban
```

```
2 gaoli liuban
```

```
3 zhanghai qiban
```

这个例子就解决了刚开始提出的那个问题，通过这个视图可以很好地保护基本表中的数据。这个视图中的信息很简单，只包含了 `id`、姓名和班级，`id` 字段对应 `student` 表中的 `s_id` 字段，`name` 字段对应 `student` 表中的 `name` 字段，`glass` 字段对应 `stu_info` 表中的 `glass` 字段。

8.2.4 创建视图的视图

Oracle 也可以在视图上创建视图。下面通过案例来学习创建的方法。

【例 8.4】在视图 `stu_glass` 上创建视图 `stu_gl_glass`，代码如下：

```
CREATE OR REPLACE VIEW stu_gl_glass
AS
SELECT stu_glass_id, stu_glass.name
FROM stu_glass;
```

查询视图 `stu_gl_glass`，SQL 语句如下：

```
SQL> SELECT * FROM stu_gl_glass;
```

```
ID NAME
```

```
-----
```

```
1 wanglin1
```

```
2 gaoli
```



3 zhanghai

从结果可以看出，视图 `stu_gl_glass` 就是把视图 `stu_glass` 中的 `GLASS` 字段去掉了。

8.2.5 创建没有源表的视图

默认情况下，如果没有源表，创建视图是会出现错误的。

【例 8.5】创建没有源表的视图，代码如下：

```
CREATE OR REPLACE VIEW gl_glass
AS
SELECT stu_glass_id, stu_glass.name
FROM glass;
```

提示错误信息如下：

错误报告：

SQL 错误：ORA-00942：表或视图不存在

说明视图创建失败。此时需要使用 `FORCE` 关键词，从而避免这种错误。

【例 8.6】强制创建没有源表的视图，代码如下：

```
CREATE OR REPLACE FORCE VIEW gl_glass
AS
SELECT stu_glass_id, stu_glass.name
FROM glass;
```

代码执行后，出现以下提示信息，说明视图已经成功创建。

错误报告：

SQL 命令：force view GL_GLASS

失败：ORA-24344：成功，但出现编译错误

8.3 查看视图

查看视图是查看数据库中已存在的视图的定义。`DESCRIBE` 可以用来查看视图，具体的语法如下：

```
DESCRIBE 视图名;
```

【例 8.7】通过 `DESCRIBE` 语句查看视图 `view_t` 的定义，代码如下：



```
DESCRIBE view_t;
```

代码执行如下：

```
SQL> DESCRIBE view_t;
```

```
DESCRIBE view_t
```

名称	空值	类型
----	----	----

-----	--	-----
-------	----	-------

QUANTITY		NUMBER(9)
----------	--	-----------

PRICE		NUMBER(9)
-------	--	-----------

结果显示出了视图的字段定义、字段的数据类型、是否为空。

DESCRIBE 一般情况下都简写成 DESC，输入这个命令的执行结果和输入 DESCRIBE 的执行结果是一样的。

8.4 修改视图

修改视图是指修改数据库中存在的视图，当基本表的某些字段发生变化时，可以通过修改视图来保持与基本表的一致性。Oracle 中通过 CREATE OR REPLACE VIEW 语句和 ALTER 语句来修改视图的约束。

8.4.1 CREATE OR REPLACE VIEW 语句修改视图

Oracle 中如果要修改视图，使用 CREATE OR REPLACE VIEW 语句，语法如下：

```
CREATE OR REPLACE [[NO]FORCE] VIEW
    [schema.] view
    [(alias,...)inline_constraint(s)]
    [out_of_line_constraint (s)]
AS subquery
[
    WITH{READ ONLY CHECK OPTION[CONSTRAINT constraint]}
];
```

可以看到，修改视图的语句和创建视图的语句是完全一样的。当视图已经存在时，修改语句对视图进行修改；当视图不存在时，创建视图。下面通过一个实例来说明。

【例 8.8】修改视图 view_t，代码如下：



```
CREATE OR REPLACE VIEW view_t AS SELECT * FROM t;
```

首先通过 DESCRIBE 查看一下更改之前的视图，以便与更改之后的视图进行对比。执行的结果如下：

```
SQL> DESCRIBE view_t;
```

```
DESCRIBE view_t
```

名称	空值	类型
----	----	----

-----	--	-----
-------	----	-------

QUANTITY		NUMBER(9)
----------	--	-----------

PRICE		NUMBER(9)
-------	--	-----------

修改视图的 SQL 语句如下：

```
SQL> CREATE OR REPLACE VIEW view_t(quty,pri) AS SELECT * FROM t;
```

查看修改后的视图，SQL 语句如下：

```
SQL> DESCRIBE view_t;
```

```
DESCRIBE view_t
```

名称	空值	类型
----	----	----

-----	--	-----
-------	----	-------

QUTY		NUMBER(9)
------	--	-----------

PRI		NUMBER(9)
-----	--	-----------

从执行的结果来看，相比原来的视图 view_t，新的视图 view_t 的字段名称被修改了。

8.4.2 ALTER 语句修改视图的约束

ALTER 语句是 Oracle 提供的另外一种修改视图约束的方法。

【例 8.9】使用 ALTER 语句为视图 view_t 添加唯一约束，代码如下：

```
ALTER VIEW view_t
```

```
ADD CONSTRAINT T_UNQ UNIQUE (QUTY)
```

```
DISABLE NOVALIDATE;
```

上面实例为字段 QUTY 添加了唯一约束，约束名称为 T_UNQ。其中 DISABLE NOVALIDATE 表示此前数据和以后数据都不检查。

另外，使用 ALTER 语句还可以删除添加的视图约束。



【例 8.10】使用 ALTER 语句删除视图 view_t 的唯一约束，代码如下：

```
ALTER VIEW view_t  
DROP CONSTRAINT T_UNQ;
```

结果提示视图已经更新，表示视图 view_t 的唯一约束已经被成功删除。

8.5 更新视图

更新视图是指通过视图来插入、更新、删除表中的数据，因为视图是一个虚拟表，其中没有数据。通过视图更新的时候都是转到基本表上进行更新的，如果对视图增加或者删除记录，实际上是对其基本表增加或者删除记录。本节将介绍视图更新的 3 种方法：INSERT、UPDATE 和 DELETE。

【例 8.11】使用 UPDATE 语句更新视图 view_t，代码如下：

```
UPDATE view_t SET qty=5;
```

执行视图更新之前，查看基本表和视图的信息，执行结果如下：

```
SQL> SELECT * FROM view_t;
```

QUTY	PRI
3	50

```
SQL> SELECT * FROM t;
```

QUANTITY	PRICE
3	50

使用 UPDATE 语句更新视图 view_t，执行过程如下：

```
SQL> UPDATE view_t SET qty=5;
```

查看视图更新之后，基本表和视图的内容：

```
SQL> SELECT * FROM t;
```

QUANTITY	PRICE
5	50

```
SQL> SELECT * FROM view_t;
```

```
QTY    PRI
```

```
-----
```

5	50
---	----

```
SQL>SELECT * FROM view_t2;
```

```
QTY    PRICE
```

```
-----
```

5	50
---	----

对视图 view_t 更新后，基本表 t 的内容也更新了，同样当对基本表 t 更新后，另外一个视图 view_t2 中的内容也会更新。

【例 8.12】使用 INSERT 语句在基本表 t 中插入一条记录，代码如下：

```
INSERT INTO t VALUES (3,5);
```

查询视图 view_t2 的内容是否更新，执行结果如下：

```
SQL> SELECT * FROM view_t2;
```

```
QTY    PRICE
```

```
-----
```

5	50
3	5

向表 t 中插入一条记录，通过 SELECT 查看表 t 和视图 view_t2，可以看到其中的内容也跟着更新。

【例 8.13】使用 DELETE 语句删除视图 view_t2 中的一条记录，代码如下：

```
DELETE FROM view_t2 WHERE price=5;
```

查询视图 view_t2 的内容是否更新，结果如下：

```
SQL> SELECT * FROM view_t2;
```

```
QTY    PRICE
```

```
-----
```

5	50
---	----

查询数据表 t 的内容是否更新，结果如下：


```
SQL> SELECT * FROM t;
```

```
QUANTITY    PRICE
```

```
-----
```

```
5           50
```

在视图 `view_t2` 中删除 `price=5` 的记录，视图中的删除操作最终是通过删除基本表中相关的记录实现的，查看删除操作之后的表 `t` 和视图 `view_t2`，可以看到通过视图删除其所依赖的基本表中的数据。

8.6 删除视图

当视图不再需要时，可以将其删除，删除一个或多个视图可以使用 `DROP VIEW` 语句，语法如下：

```
DROP VIEW view_name
```

其中，`view_name` 是要删除的视图名称。删除视图必须拥有 `DROP` 权限。

【例 8.14】删除 `stu_glass` 视图，代码如下：

```
DROP VIEW stu_glass;
```

执行结果：

```
SQL> DROP VIEW IF EXISTS stu_glass;
```

```
view STU_GLASS 已删除。
```

如果名称为 `stu_glass` 的视图存在，该视图将被删除。使用 `DESC VIEW` 语句查看操作结果：

```
SQL> DESC VIEW stu_glass;
```

```
ERROR:
```

```
-----
```

```
错误：对象 VIEW 不存在
```

可以看到，`stu_glass` 视图已经不存在，删除成功。

8.7 限制视图的数据操作

对视图数据的增加或更新实际上是操作视图的源表。通过对视图的限制操作，可以提高数据操

作安全性。

8.7.1 设置视图的只读属性

如果想防止用户修改数据，可以将视图设成只读属性。

【例 8.15】在 t 表格上创建一个名为 view_tt 的只读视图，代码如下：

```
CREATE OR REPLACE VIEW view_tt AS
SELECT quantity, price FROM t
WITH READ ONLY;
```

创建完成后，如果向视图 view_tt 插入、更新和删除数据时，会提示错误信息。

8.7.2 设置视图的检查属性

在修改视图的数据时，可以指定一定的检查条件。此时需要使用 WITH CHECK OPTION 来设置视图的检查属性，表示启动了和子查询条件一样的约束。

【例 8.16】在 t 表格上创建一个名为 view_tc 的视图，限制条件为字段 price 的值大于 10，代码如下：

```
CREATE OR REPLACE VIEW view_tc AS
SELECT quantity, price FROM t
WHERE price>10
WITH CHECK OPTION;
```

创建完成后，向视图 view_tc 插入、更新和删除数据时，会受到检查条件的限制。

【例 8.17】向视图 view_tc 插入数据(3,5)，代码如下：

```
INSERT INTO view_tc VALUES (3,5);
```

错误报告：

```
SQL 错误: ORA-01402: 视图 WITH CHECK OPTION where 子句违规
```

这里添加的 price 的值小于 10，所以出现错误提示。同样，更新和删除操作也受到限制条件的约束。

8.8 综合案例——视图应用

本章介绍了 Oracle 数据库中视图的含义和作用，并且讲解了创建视图、修改视图和删除视图的方法。创建视图和修改视图是本章的重点。这两部分的内容比较多，而且比较复杂，希望读者能够认真学习这两部分的内容，并且在计算机上进行操作。读者在创建视图之后一定要查看视图的结

构，确保创建的视图是正确的，修改过视图后也要查看视图的结构，保证修改是正确的。

1. 案例目的

掌握视图的创建、查询、更新和删除操作。

假如 Henan、Hebei 的 3 个学生参加 Tsinghua University、Peking University 的自学考试，现在需要用数据对其考试的结果进行查询和管理，Tsinghua University 的分数线为 40，Peking University 的分数线为 41。学生表包含学生的学号、姓名、家庭地址和电话号码；报名表包含学号、姓名、所在学校和报名的学校，表结构以及表中的内容分别如表 8-1~表 8-6 所示。

表 8-1 stu 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
s_id	NUMBER(11)	是	否	是	是	否
s_name	VARCHAR2(20)	否	否	是	否	否
addr	VARCHAR2(50)	否	否	是	否	否
tel	VARCHAR2(50)	否	否	是	否	否

表 8-2 sign 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
s_id	NUMBER(11)	是	否	是	是	否
s_name	VARCHAR2(20)	否	否	是	否	否
s_sch	VARCHAR2(50)	否	否	是	否	否
s_sign_sch	VARCHAR2(50)	否	否	是	否	否

表 8-3 stu_mark 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
s_id	NUMBER(11)	是	否	是	是	否
s_name	VARCHAR2(20)	否	否	是	否	否
mark	NUMBER(11)	否	否	是	否	否

表 8-4 stu 表内容

s_id	s_name	addr	tel
1	XiaoWang	Henan	0371-12345678
2	XiaoLi	Hebei	13889072345
3	XiaoTian	Henan	0371-12345670

表 8-5 sign 表内容

s_id	s_name	s_sch	s_sign_sch
1	XiaoWang	Middle School1	Peking University
2	XiaoLi	Middle School2	Tsinghua University
3	XiaoTian	Middle School3	Tsinghua University

表 8-6 stu_mark 表内容

s_id	s_name	mark
1	XiaoWang	80
2	XiaoLi	71
3	XiaoTian	70

2. 案例操作过程

步骤 01 创建学生表 stu，插入 3 条记录。

创建学生表和插入数据代码如下：

```
CREATE TABLE stu
(
s_id NUMBER (11) PRIMARY KEY,
s_name VARCHAR2 (20) NOT NULL,
addr VARCHAR2 (50) NOT NULL,
tel VARCHAR2 (50) NOT NULL
);
INSERT INTO stu VALUES
(1, 'XiaoWang', 'Henan', '0371-12345678') ;
INSERT INTO stu VALUES
(2, 'XiaoLi', 'Hebei', '13889072345') ;
INSERT INTO stu VALUES
(3, 'XiaoTian', 'Henan', '0371-12345670');
```

查询学生表 stu 结果如下：

```
SQL> SELECT * FROM stu;
```

S_ID	S_NAME	ADDR	TEL
1	XiaoWang	Henan	0371-12345678
2	XiaoLi	Hebei	13889072345
3	XiaoTian	Henan	0371-12345670

通过上面的代码执行后，在当前的数据库中创建了一个表 stu，通过插入语句向表 stu 中插入了 3 条记录。stu 表的主键为 s_id。

步骤 02 创建报名表 sign，插入 3 条记录。

创建报名表并插入数据，代码如下：

```
CREATE TABLE sign
(
s_id NUMBER (11) PRIMARY KEY,
```



```

s_name VARCHAR2(20) NOT NULL,
s_sch VARCHAR2(50) NOT NULL,
s_sign_sch VARCHAR2(50) NOT NULL
);
INSERT INTO sign VALUES
(1,'XiaoWang','Middle School1','Peking University') ;
INSERT INTO sign VALUES
(2,'XiaoLi','Middle School2','Tsinghua University') ;
INSERT INTO sign VALUES
(3,'XiaoTian','Middle School3','Tsinghua University');

```

查询报名表 sign 结果如下：

```
SQL> SELECT * FROM sign;
```

S_ID	S_NAME	S_SCH	S_SIGN_SCH
1	XiaoWang	Middle School1	Peking University
2	XiaoLi	Middle School2	Tsinghua University
3	XiaoTian	Middle School3	Tsinghua University

创建一个 sign 表，同时向表中插入了 3 条报考记录。

步骤 03 创建成绩表 stu_mark，插入 3 条记录。

创建成绩表，代码如下：

```

CREATE TABLE stu_mark
(
s_id NUMBER (11) PRIMARY KEY ,
s_name VARCHAR2(20) NOT NULL,
mark NUMBER (11) NOT NULL
);
INSERT INTO stu_mark VALUES
(1,'XiaoWang',80) ;
INSERT INTO stu_mark VALUES
(2,'XiaoLi',71) ;
INSERT INTO stu_mark VALUES

```



```
(3, 'XiaoTian', 70);
```

查询成绩表 `stu_mark`，结果如下：

```
SQL> SELECT * FROM stu_mark;
```

```
S_ID  S_NAME    MARK
```

```
-----
```

```
1  XiaoWang    80
```

```
2  XiaoLi      71
```

```
3  XiaoTian    70
```

创建 `stu_mark` 表，向学生的成绩表插入 3 条成绩记录。

步骤 04 创建考上 Peking University 的学生的视图。

创建考上 Peking University 的学生的视图，代码如下：

```
CREATE VIEW beida (id,name,mark,sch)
```

```
AS SELECT stu_mark.s_id,stu_mark.s_name,stu_mark.mark, sign.s_sign_sch
```

```
FROM stu_mark ,sign
```

```
WHERE      stu_mark.s_id=sign.s_id      AND      stu_mark.mark>=41      AND
```

```
sign.s_sign_sch='Peking University';
```

查询视图 `beida` 结果如下：

```
SQL> SELECT *FROM beida;
```

```
ID      NAME      MARK      SCH
```

```
-----
```

```
1  XiaoWang    80      Peking University
```

视图 `beida` 包含了考上 Peking University 的学生学号、姓名、成绩和报考的学校名称，其中，报考的学校名称为 Peking University。通过 `SELECT` 语句进行查看，可以获得成绩在 Peking University 分数线之上的学生信息。

步骤 05 创建考上 Tsinghua University 的学生的视图。

创建考上 Tsinghua University 的学生的视图，代码如下：

```
CREATE VIEW qinghua (id,name,mark,sch)
```

```
AS SELECT stu_mark.s_id, stu_mark.s_name, stu_mark.mark, sign.s_sign_sch
```

```
FROM stu_mark ,sign
```



```
WHERE      stu_mark.s_id=sign.s_id          AND      stu_mark.mark>=40      AND
sign.s_sign_sch='Tsinghua University';
```

查询视图 qinghua 结果如下：

```
SQL> SELECT * FROM qinghua ;

ID   NAME   MARK   SCH
-----
2    XiaoLi   71    Tsinghua University
3    XiaoTian  70    Tsinghua University
```

视图 qinghua 只包含了成绩在 Tsinghua University 分数线之上的学生的信息，这些信息包括学号、姓名、成绩和报考学校。

步骤 06 XiaoTian 的成绩在录入的时候多录了 50 分，对其录入成绩进行更正。

更新 XiaoTian 的成绩，代码如下：

```
UPDATE stu_mark SET mark = mark-50 WHERE stu_mark.s_name ='XiaoTian';
```

XiaoTian 的录入成绩发生了错误，当更新 XiaoTian 的成绩后，视图中是否还有 XiaoTian 被 Tsinghua University 录取的信息呢？

步骤 07 查看更新过后视图和表的情况。

查看更新后的表和视图情况，代码如下：

```
SELECT * FROM stu_mark;

SELECT * FROM qinghua;
```

执行结果如下：

```
SQL> SELECT * FROM stu_mark;

S_ID  S_NAME   MARK
-----
1    XiaoWang   80
2    XiaoLi    71
3    XiaoTian  20

SQL> SELECT * FROM qinghua ;
```

ID	NAME	MARK	SCH
2	XiaoLi	71	Tsinghua University

从结果来看，视图 qinghua 中已经不存在 XiaoTian 的信息了，说明更新成绩基本表 stu_mark 后，视图 qinghua 的内容也相应地更新了。

步骤 08 删除创建的视图。

删除 beida、qinghua 视图，执行的过程如下：

```
SQL> DROP VIEW beida;
view BEIDA 已删除。
SQL> DROP VIEW qinghua;
view QINGHUA 已删除。
```

语句执行完毕，qinghua 和 beida 两个视图分别被成功地删除。

8.9 疑难解惑

疑问 1：Oracle 中视图和表的区别以及联系是什么？

1. 两者的区别

(1) 视图是已经编译好的 SQL 语句，是基于 SQL 语句的结果集的可视化的表，而表不是。

(2) 视图没有实际的物理记录，而基本表有。

(3) 表是内容，视图是窗口。

(4) 表占用物理空间而视图不占用物理空间，视图只是逻辑概念的存在，表可以及时对它进行修改，但视图只能用创建的语句来修改。

(5) 视图是查看数据表的一种方法，可以查询数据表中某些字段构成的数据，只是一些 SQL 语句的集合。从安全的角度来说，视图可以防止用户接触数据表，因而用户不知道表结构。

(6) 表属于全局模式的表，是实表；视图属于局部模式的表，是虚表。

(7) 视图的建立和删除只影响视图本身，不影响对应的基本表。

2. 两者的联系

视图 (View) 是在基本表之上建立的表，它的结构 (即所定义的列) 和内容 (即所有记录) 都来自基本表，它依据基本表存在而存在。一个视图可以对应一个基本表，也可以对应多个基本表。

视图是基本表的抽象和在逻辑意义上建立的新关系。

疑问 2：什么时候视图不能做更新操作？

当视图中包含如下内容时，视图的更新操作将不能被执行：

- (1) 视图中不包含基表中被定义为非空的列。
- (2) 在定义视图的 SELECT 语句后的字段列表中使用了数学表达式。
- (3) 在定义视图的 SELECT 语句后的字段列表中使用了聚合函数。
- (4) 在定义视图的 SELECT 语句中使用了 DISTINCT, UNION, TOP, GROUP BY 或 HAVING 子句。

8.10 经典习题

- (1) 如何在一个表上创建视图？
- (2) 如何在多个表上建立视图？
- (3) 如何更改视图？
- (4) 如何查看视图的详细信息？
- (5) 如何更新视图的内容？
- (6) 如何理解视图和基本表之间的关系、用户操作的权限？

第 9 章

◀ 游 标 ▶

查询语句可能返回多条记录，如果数据量非常大，需要使用游标来逐条读取查询结果集中的记录。应用程序可以根据需要滚动或浏览其中的数据。本章将介绍游标的概念、游标的分类、游标的基本操作等内容。

- 了解游标的基本概念
- 掌握显式游标的使用方法
- 掌握隐式游标的使用方法
- 掌握综合案例中游标的操作技巧和方法

9.1 认识游标

游标是 Oracle 的一种数据访问机制，它允许用户访问单独的数据行，用户可以对每一行进行单独处理，从而降低系统开销和潜在的阻隔情况，用户也可以使用这些数据生成 SQL 代码并立即执行或输出。

9.1.1 游标的概念

游标类似一个可以变动的光标。类似于 C 语言中的指针，它可以指向结果集中的任意位置。在查看或处理结果集中的数据时，游标可以提供在结果集中向前或向后浏览数据的功能。当要对结果集进行逐行单独处理时，必须声明一个指向该结果集的游标变量。游标默认指向的是结果集的首记录。

默认情况下，游标可以返回当前执行的行记录，只能返回一行记录。如果想要返回多行，需要不断地滚动游标，把需要的数据查询一遍。用户可以操作游标所在位置行的记录，例如把返回记录作为另一个查询的条件等。

9.1.2 游标的优点

SELECT 语句返回的是一个结果集，但有的时候应用程序并不总是能对整个结果集进行有效的处理，游标便提供了这样一种机制，它能从包括多条数据记录的结果集中每次提取一条记

录，游标总是与一条 SQL 选择语句相关联，由结果集和指向特定记录的游标位置组成。使用游标具有以下优点。

- (1) 允许程序对由 SELECT 查询语句返回的行集中的每一行执行相同或不同的操作，而不是对整个集合执行同一个操作。
- (2) 提供对基于游标位置的表中的行进行删除和更新的能力。
- (3) 游标作为数据库管理系统和应用程序设计之间的桥梁，将两种处理方式连接起来。

9.1.3 游标的分类

Oracle 中游标分为静态游标和 REF 游标两类。本章只对常用的静态游标作详细介绍。静态游标分为两种类型：显式游标和隐式游标。

(1) 显式游标：在使用之前必须有明确的游标声明和定义，这样的游标定义会关联数据查询语句，通常会返回一行或多行。打开游标后，用户可以利用游标的位置对结果集进行检索，使之返回单一的行记录，用户可以操作此记录。关闭游标后，就不能再对结果集进行任何操作。显式游标需要用户自己写代码完成，一切由用户控制。

(2) 隐式游标：隐式游标和显式游标不同，它被数据库自动管理，此游标用户无法控制，但能得到它的属性信息。

9.2 显式游标

介绍完游标的概念和分类等内容之后，下面将向各位读者介绍如何操作显式游标，对于显式游标的操作主要有以下内容：声明游标、打开游标、读取游标中的数据和关闭游标。

9.2.1 显式游标的语法

使用游标之前，要声明游标。声明显式游标的语法如下：

```
CURSOR cursor_name
    [(parameter_name datatype,...)]
    IS select_statement;
```

其中 CURSOR 表示声明游标；cursor_name 是游标的名称；parameter_name 表示参数名称；datatype 表示参数类型；select_statement 是游标关联的 SELECT 语句。

【例 9.1】声明名称为 cursor_fruit 的游标，输入语句如下：

```
DECLARE CURSOR cursor_fruit
IS SELECT f_name, f_price FROM fruits ;
```

上面的代码中，定义游标的名称为 `cursor_fruit`，`SELECT` 语句表示从 `fruits` 表中查询出 `f_name` 和 `f_price` 字段的值。

9.2.2 打开游标

在使用游标之前，必须打开游标。打开游标的语法格式如下：

```
OPEN cursor_name ;
```

【例 9.2】打开上例中声明的名称为 `cursor_fruit` 的游标，输入语句如下：

```
OPEN cursor_fruit ;
```

9.2.3 读取游标中的数据

打开游标之后，就可以读取游标中的数据了，`FETCH` 命令可以读取游标中的某一行数据。`FETCH` 语句的语法格式如下：

```
FETCH cursor_name INTO Record_name;
```

读取的记录放到变量当中。如果想让 `FETCH` 读取多个记录，`FETCH` 需要和循环语句一起使用，直到某个条件不符合要求而退出。使用 `FETCH` 时游标属性 `%ROWCOUNT` 会不断累加。

【例 9.3】使用名称为 `cursor_fruit` 的游标，检索 `fruits` 表中的记录，输入语句如下：

```
FETCH cursor_fruit INTO Record_name;
```

9.2.4 关闭游标

打开游标以后，服务器会专门为游标开辟一定的内存空间存放游标操作的数据结果集合，同时游标的使用也会根据具体情况对某些数据进行封锁。所以在不使用游标的时候，可以将其关闭，以释放游标所占用的服务器资源。关闭游标使用 `CLOSE` 语句，语法格式如下：

```
CLOSE cursor_name
```

【例 9.4】关闭名称为 `cursor_fruit` 的游标，输入语句如下：

```
CLOSE cursor_fruit;
```

9.2.5 使用显式游标的案例

下面通过一个案例来学习显式游标的整个过程。

【例 9.5】定义名称为 `frt_cur` 的游标，然后打开、读取和关闭游标 `frt_cur`。输入语句如下：

```
set serveroutput on;

DECLARE
```



```

CURSOR frt_cur

IS SELECT f_id,f_name FROM fruits ;

cur_fruits frt_cur%ROWTYPE;

BEGIN

    OPEN frt_cur;

        FETCH frt_cur INTO cur_fruits;

        dbms_output.put_line(cur_fruits.f_id||'.'||cur_fruits.f_name);

CLOSE frt_cur;

END;

```

上述代码的具体含义如下：

- set serveroutput on: 打开 Oracle 自带的输出方法 dbms_output。
- CURSOR frt_cur: 声明一个名称为 frt_cur 的游标。
- IS SELECT f_id,f_name FROM fruits: 表示游标关联的查询。
- cur_fruits frt_cur%ROWTYPE: 定义一个游标变量，名称为 cur_fruits。
- OPEN frt_cur: 表示打开游标。
- FETCH frt_cur INTO cur_fruits: 表示利用 FETCH 语句从结果集中提取指针指向的当前行记录。
- dbms_output.put_line(cur_fruits.f_id||'.'||cur_fruits.f_name): 表示输出结果并换行，这里输出表 fruits 中的 f_id 和 f_name 两个字段的值。

在 Oracle SQL Developer 中运行上面的代码，执行结果如下：

```
a1.apple
```

通过上面的案例，读者可以充分理解显式游标的 4 个基本步骤。

9.2.6 使用显式游标的 LOOP 语句

上一节中的案例只提取一条数据，如果用户想使用显式游标提取多条记录，就需要一个遍历结果集的方法，这就是 LOOP 语句的作用。

【例 9.6】通过 LOOP 语句遍历游标，输入语句如下：

```

set serveroutput on;

DECLARE

CURSOR frt_loop_cur

```

```

IS SELECT f_id,f_name,f_price FROM fruits
WHERE f_price>10;

cur_id fruits.f_id%TYPE;
cur_name fruits.f_name%TYPE;
cur_price fruits.f_name%TYPE;

BEGIN
    OPEN frt_loop_cur;
    LOOP
        FETCH frt_loop_cur INTO cur_id,cur_name,cur_price;
        EXIT WHEN frt_loop_cur%NOTFOUND;
        dbms_output.put_line(cur_id||'.'||cur_name
                            ||'.'||cur_price);
    END LOOP;
    CLOSE frt_loop_cur;
END;

```

其中，某些代码的具体含义如下：

- `cur_id fruits.f_id%TYPE`: 表示变量类型同表 `fruits` 的对应的字段类型一致。
- `EXIT WHEN frt_loop_cur%NOTFOUND`: 表示利用游标的属性实现没有记录时退出循环。

在 Oracle SQL Developer 中运行上面的代码，执行结果如下：

```

b1.blackberry.10.2
bs1.orange.11.2
t1.banana.10.3
m1.mango.15.6
m3.xxtt.11.6

```

案例中通过使用 `LOOP` 语句，把所有符合条件的记录全部输出。

9.2.7 使用 BULK COLLECT 和 FOR 语句的游标

使用 `FETCH...INTO...` 语句只能提取单条数据。如果在数据比较大的情况下，执行效率就比较低。为了解决这一问题，可以使用 `FETCH...BULK COLLECT INTO...` 语句批量提取数据。

【例 9.7】通过 `BULK COLLECT` 和 `FOR` 语句遍历游标，输入语句如下：


```

set serveroutput on;

DECLARE

CURSOR frt_collect_cur
IS SELECT * FROM fruits
WHERE f_price>10;

TYPE FRT_TAB IS TABLE OF FRUITS%ROWTYPE;

frt_rd FRT_TAB;

BEGIN

    OPEN frt_collect_cur;

    LOOP

        FETCH frt_collect_cur BULK COLLECT INTO frt_rd LIMIT 2;

        FOR i in 1..frt_rd.count LOOP

            dbms_output.put_line(frt_rd(i).f_id||'.'||frt_rd(i).f_name
                                ||'.'||frt_rd(i).f_price);

        END LOOP;

        EXIT WHEN frt_collect_cur%NOTFOUND;

    END LOOP;

    CLOSE frt_collect_cur;

END;

```

其中, 以下代码是定义和表 `fruits` 行对象一致的集合类型 `frt_rd`, 该变量用于存放批量得到的数据。

```

TYPE FRT_TAB IS TABLE OF FRUITS%ROWTYPE;

frt_rd FRT_TAB;

```

LIMIT 2 表示每次提取两条。

在 Oracle SQL Developer 中运行上面的代码, 执行结果如下:

```

b1.blackberry.10.2
bs1.orange.11.2
t1.banana.10.3
m1.mango.15.6
m3.xxxtt.11.6

```

9.2.8 使用 CURSOR FOR LOOP 语句的游标

通过使用 CURSOR FOR LOOP 语句, 可以在不声明变量的情况下, 提取数据。



【例 9.8】通过 CURSOR FOR LOOP 语句遍历游标，输入语句如下：

```
set serveroutput on;

DECLARE

CURSOR frt1 IS SELECT * FROM fruits
WHERE f_price<10;

BEGIN

    FOR curfrt1 IN frt1

        LOOP

            dbms_output.put_line(curfrt1.f_id||'.'|| curfrt1.f_name
                                ||'.'|| curfrt1.f_price);

        END LOOP;

END;
```

在 Oracle SQL Developer 中运行上面的代码，执行结果如下：

```
a1.apple.5.2
bs2.melon.8.2
t2.grape.5.3
o2.coconut.9.2
c0.cherry.3.2
a2.apricot.2.2
l2.lemon.6.4
b2.berry.7.6
m2.xbabay.2.6
t4.xbababa.3.6
b5.xxxx.3.6
```

9.2.9 显式游标的属性

利用游标属性可以得到游标执行的相关信息。显式游标有以下 4 个属性：

- %ISOPEN: 用于判断游标属性是否打开。如果打开则返回 TRUE，否则返回 FALSE。
- %FOUND: 用于检查行数据是否有效。如果有效则返回 TRUE，否则返回 FALSE。
- %NOTFOUND: 与 %FOUND 属性相反。如果没有提取出数据则返回 TRUE，否则返回 FALSE。
- %ROWCOUNT: 表示累计到当前为止使用 FETCH 提取数据的行数。

【例 9.9】通过 %ISOPEN 属性判断游标是否打开，输入语句如下：



```
set serveroutput on;

DECLARE

CURSOR frt2 IS SELECT * FROM fruits;

cur_fruits fruits%ROWTYPE;

BEGIN

    IF frt2%ISOPEN THEN

        FETCH frt2 INTO cur_fruits;

        dbms_output.put_line(cur_fruits.f_id||'.'|| cur_fruits.f_name

                               ||'.'|| cur_fruits.f_price);

    ELSE

        dbms_output.put_line('游标 frt2 没有打开');

    END IF;

END;
```

在 Oracle SQL Developer 中运行上面的代码，执行结果如下：

游标 frt2 没有打开

【例 9.10】通过%FOUND 属性判断数据的有效性，输入语句如下：

```
set serveroutput on;

DECLARE

CURSOR frt_found_cur

IS SELECT * FROM fruits;

cur_prodrdc FRUITS%ROWTYPE;

BEGIN

    OPEN frt_found_cur;

    LOOP

        FETCH frt_found_cur INTO cur_prodrdc;

        IF frt_found_cur%FOUND THEN

            dbms_output.put_line(cur_prodrdc.f_id||'.'|| cur_prodrdc.f_name

                                   ||'.'|| cur_prodrdc.f_price);

        ELSE

            dbms_output.put_line('没有数据被提取');

            EXIT;

        END IF;

    END LOOP;

END;
```




```
END LOOP;  
CLOSE frt_found_cur;  
END;
```

在 Oracle SQL Developer 中运行上面的代码, 执行结果如下:

```
a1.apple.5.2  
b1.blackberry.10.2  
bs1.orange.11.2  
bs2.melon.8.2  
t1.banana.10.3  
t2.grape.5.3  
o2.coconut.9.2  
c0.cherry.3.2  
a2.apricot.2.2  
l2.lemon.6.4  
b2.berry.7.6  
m1.mango.15.6  
m2.xbabay.2.6  
t4.xbababa.3.6  
m3.xxtt.11.6  
b5.xxxx.3.6  
没有数据被提取
```

%NOTFOUND 属性的含义与%FOUND 属性正好相反, 这里就不作讲解。

【例 9.11】通过%ROWCOUNT 属性查看已经返回了多少行记录, 输入语句如下:

```
set serveroutput on;  
  
DECLARE  
CURSOR frt_rowcount_cur  
IS SELECT * FROM fruits  
WHERE f_price<8;  
  
TYPE FRT_TAB IS TABLE OF FRUITS%ROWTYPE;  
frt_count_rd FRT_TAB;  
  
BEGIN
```




```
OPEN frt_rowcount_cur;
LOOP
    FETCH frt_rowcount_cur BULK COLLECT INTO frt_count_rd LIMIT 2;
    FOR i in frt_count_rd.first..frt_count_rd.last LOOP
dbms_output.put_line(frt_count_rd(i).f_id||'. '
|| frt_count_rd(i).f_name||'. '|| frt_count_rd(i).f_price);
    END LOOP;
    IF mod(frt_rowcount_cur%ROWCOUNT,2)=0 THEN
dbms_output.put_line('读取到第'||frt_rowcount_cur%ROWCOUNT||'
条记录');
    ELSE
dbms_output.put_line('读取到单条记录为'
||frt_rowcount_cur%ROWCOUNT||'条记录');
    END IF;
    EXIT WHEN frt_rowcount_cur%NOTFOUND;
    END LOOP;
    CLOSE frt_rowcount_cur;
END;
```

在 Oracle SQL Developer 中运行上面的代码，执行结果如下：

```
a1.apple.5.2
t2.grape.5.3
读取到第2
条记录
c0.cherry.3.2
a2.apricot.2.2
读取到第4
条记录
l2.lemon.6.4
b2.berry.7.6
读取到第6
条记录
m2.xbabay.2.6
t4.xbababa.3.6
```



读取到第8

条记录

b5.xxxx.3.6

读取到单条记录为9条记录

9.3 隐式游标

上一小节中向各位读者介绍了显式游标的使用方法和技巧,本小节将对隐式游标的使用方法进行进一步的介绍,包括使用隐式游标、隐式游标的属性和如何在游标中使用异常处理。

9.3.1 使用隐式游标

隐式游标是由数据库自动创建和管理的游标,默认名称为 SQL,也称为 SQL 游标。

每当运行 SELECT 语句时,系统会自动打开一个隐式的游标,用户不能控制隐式游标,但是可以使用隐式游标。

【例 9.12】使用隐式游标,输入语句如下:

```
set serveroutput on;

DECLARE

cur_id fruits.f_id%TYPE;
cur_name fruits.f_name%TYPE;
cur_price fruits.f_name%TYPE;
BEGIN
SELECT f_id,f_name,f_price INTO cur_id,cur_name,cur_price
FROM fruits
WHERE f_price=5.3;
IF SQL%FOUND THEN
    dbms_output.put_line(cur_id||'.'||cur_name||'.'||cur_price);
END IF;
END;
```

在 Oracle SQL Developer 中运行上面的代码,执行结果如下:

t2.grape.5.3

上面代码中的判断条件如下:



```
WHERE f_price=5.3;
```

必须保证只有一条记录符合，因为 SELECT INTO 语句只能返回一条记录，如果返回多条记录，在 Oracle SQL Developer 中运行时，会提示：实际返回的行数超过请求的行数。

9.3.2 隐式游标的属性

隐式游标的属性种类和显式游标是一样的，但是属性的含义有一定的区别。

- %ISOPEN: Oracle 自行控制该属性，返回的永远是 FALSE。
- %FOUND: 该属性反映操作是否影响了数据，如果影响了数据，返回 TRUE，否则返回 FALSE。
- %NOTFOUND: 与 %FOUND 属性相反。如果操作没有影响数据则返回 TRUE，否则返回 FALSE。
- %ROWCOUNT: 该属性反映操作对数据影响的数量。

【例 9.13】验证隐式游标的 ISOPEN 属性返回值为 FALSE 的特性，输入语句如下：

```
set serveroutput on;

DECLARE

BEGIN

    DELETE FROM fruits;

    IF SQL%ISOPEN THEN

        dbms_output.put_line('游标打开了');

    ELSE

        dbms_output.put_line('游标没有打开');

    END IF;

END;
```

在 Oracle SQL Developer 中运行上面的代码，执行结果如下：

```
游标没有打开
```

%FOUND 属性在 INSERT、UPDATE 和 DELETE 执行对数据有影响时会返回 TRUE，而 SELECT INTO 语句只要语句返回，该属性即为 TRUE。

【例 9.14】隐式游标属性 %FOUND 的应用，输入语句如下：

```
set serveroutput on;

DECLARE

    cur_id fruits.f_id%TYPE;

    cur_name fruits.f_name%TYPE;
```



```
cur_price fruits.f_price%TYPE;
BEGIN
    SELECT f_id ,f_name,f_price INTO cur_id,cur_name,cur_price
    FROM fruits;

    EXCEPTION
    WHEN TOO_MANY_ROWS THEN
    IF SQL%FOUND THEN
        dbms_output.put_line('%FOUND 为 TRUE');
        DELETE FROM fruits WHERE f_price=100.2;
    IF SQL%FOUND THEN
        dbms_output.put_line('删除数据了');
    END IF;
    END IF;
END;
```

以下代码的含义是当返回多条数据时会出现 TOO_MANY_ROWS 异常, 执行 THEN 后面的脚本, 这是对可能引起的异常的处理。

```
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
```

以下代码表示当 SQL%FOUND 为 TURE 时, 执行删除操作。

```
DELETE FROM fruits WHERE f_price=100.2;
```

以下代码表示继续判断 SQL%FOUND 是否为 TURE, 如果是 TURE, 则继续 THEN 后的操作。

```
IF SQL%FOUND THEN
    dbms_output.put_line('删除数据了');
```

在 Oracle SQL Developer 中运行上面的代码, 执行结果如下:

```
%FOUND 为 TRUE
```

从结果可以看出该属性的使用方法和特征。由于在删除操作时没有在数据库中找到符合 WHERE 条件的记录, 所以没有删除操作, 此时的 SQL%FOUND 为 FALSE, 后面的删除提示没有执行。

%NOTFOUND 属性的含义与%FOUND 属性正好相反, 这里就不作讲解。

【例 9.15】通过%ROWCOUNT 属性查看已经返回了多少行记录, 输入语句如下:



```
set serveroutput on;

DECLARE
    cur_id fruits.f_id%TYPE;
    cur_name fruits.f_name%TYPE;
    cur_price fruits.f_price%TYPE;
    cur_count varchar2(8);
BEGIN
    SELECT f_id ,f_name,f_price INTO cur_id,cur_name,cur_price
    FROM fruits;

    EXCEPTION

    WHEN NO_DATA_FOUND THEN

        dbms_output.put_line('SQL%ROWCOUNT');
        dbms_output.put_line('没有数据');

    WHEN TOO_MANY_ROWS THEN

        cur_count:= SQL%ROWCOUNT;

        dbms_output.put_line(' SQL%ROWCOUNT 值为: '||cur_count);
END;
```

在 Oracle SQL Developer 中运行上面的代码，执行结果如下：

```
SQL%ROWCOUNT 值为: 1
```

通过结果可知，定义变量 `cur_count` 保存 `SQL%ROWCOUNT` 是成功的。

9.3.3 在游标中使用异常处理

从上一节例子可以看出，当出现异常情况时，用户可以提前做好处理操作。如果不加处理，则脚本会中断操作。可见，合理地处理异常，可以维护脚本运行的稳定性。

【例 9.16】在游标中使用异常处理。

为了演示效果，可以先将 `fruits` 表中的数据删除，SQL 语句如下：

```
delete fruits;
```

针对没有数据的异常处理代码如下：

```
set serveroutput on;
```



```
DECLARE

    cur_id fruits.f_id%TYPE;
    cur_name fruits.f_name%TYPE;
BEGIN

    SELECT f_id ,f_name INTO cur_id,cur_name
    FROM fruits;

    EXCEPTION

    WHEN NO_DATA_FOUND THEN

        dbms_output.put_line('没有数据');

END;
```

在 Oracle SQL Developer 中运行上面的代码，执行结果如下：

没有数据

通过结果可知，对于没有数据的异常情况，用户提前做好了处理。

9.4 综合案例——游标的综合应用

本节将讲述一个游标的综合应用案例。通过本节的学习，读者可以更加熟练操作游标，从而解决实际工作中的问题。

1. 案例目的

案例中牵涉到两张表，分别为 fruit（水果表）和 fruitage（水果信息表）。利用游标转换两张表的数据，要求把价格高于 10 的水果放到 fruitage 中。

2. 案例操作过程

步骤 01 创建 fruit 表并插入数据。

创建 fruit 表，输入语句如下：

```
CREATE TABLE fruit
(
    f_id    varchar2(10)    NOT NULL,
```


Oracle 12c 从零开始学（视频教学版）

```
f_name varchar2(255) NOT NULL,
f_price number (8,2) NOT NULL
);
```

为了演示，需要插入如下数据：

```
INSERT INTO fruit VALUES ('a1', 'apple', 5.2);
INSERT INTO fruit VALUES ('b1', 'blackberry', 10.2);
INSERT INTO fruit VALUES ('bs1', 'orange', 11.2);
INSERT INTO fruit VALUES ('bs2', 'melon', 8.2);
INSERT INTO fruit VALUES ('t1', 'banana', 10.3);
INSERT INTO fruit VALUES ('t2', 'grape', 5.3);
INSERT INTO fruit VALUES ('o2', 'coconut', 9.2);
```

步骤 02 创建表 fruitage。

表 fruitage 和表 fruit 的字段一致，利用以下语句创建：

```
CREATE TABLE fruitage AS SELECT * FROM fruit
WHERE 2=3;
```

如果 WHERE 后面的条件为真，则复制表时把数据也一起复制。

步骤 03 创建游标，完成数据转移的要求。

```
set serveroutput on;

DECLARE
    cur_id fruit.f_id%TYPE;
    cur_name fruit.f_name%TYPE;
    cur_price fruit.f_price%TYPE;
    CURSOR frt_cur
    IS SELECT f_id ,f_name f_price INTO cur_id,cur_name,cur_price
    FROM fruit
    WHERE f_price>10;
BEGIN
    OPEN frt_cur;
    LOOP
        FETCH frt_cur INTO cur_id,cur_name,cur_price;
        IF frt_cur%FOUND THEN
```



```
INSERT INTO fruitage VALUES(cur_id, cur_name,cur_price);  
  
ELSE  
  
    dbms_output.put_line('已经取出所有数据，共有'||firt_cur%ROWCOUNT  
||'条记录');  
  
EXIT;  
  
END IF;  
  
END LOOP;  
  
CLOSE firt_cur;  
  
END;
```

在 Oracle SQL Developer 中运行上面的代码，执行结果如下：

已经取出所有数据，共有3条记录

9.5 疑难解惑

疑问 1：游标使用完后如何处理？

在使用完游标之后，一定要将其关闭。关闭游标的作用是释放游标和数据库的连接，将其从内存中删除，删除将释放系统资源。

疑问 2：执行游标后，没有输出内容，只显示“匿名块已完成”，怎么回事？

在 Oracle SQL Developer 中运行游标内容，必须在开头部分添加如下代码：

```
set serveroutput on;
```

否则，运行完成只会显示以下信息：

匿名块已完成

9.6 经典习题

- (1) 游标的含义及分类。
- (2) 使用游标的基本操作步骤都有哪些？
- (3) 打开 stu_info 表，使用游标查看 stu_info 表中成绩小于 70 的记录。



第 10 章

◀ 存储过程 ▶

简单地说，存储过程就是一条或者多条 SQL 语句的集合，可视为批文件，但是其作用不仅限于批处理。本章主要介绍如何创建存储过程，如何调用、查看、修改、删除存储过程等。

- 掌握如何创建存储过程
- 掌握如何调用存储过程
- 熟悉如何查看存储过程
- 掌握无参数和有参数的存储过程的使用方法
- 掌握修改存储过程的方法
- 熟悉如何删除存储过程
- 掌握综合案例中创建存储过程的方法和技巧

10.1 创建存储过程

在数据转换或查询报表时经常使用存储过程，它的作用是 SQL 语句不可替代的。本节主要讲述存储过程的概念和如何创建一个存储过程。

10.1.1 什么是存储过程

存储过程是指在 Oracle 数据库中，一组为了完成特定功能的 SQL 语句集，存储在数据库中经过第一次编译后再次调用不需要再次编译，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象，任何一个设计良好的数据库应用程序都应该用到存储过程。

相对于直接使用 SQL 语句，在应用程序中直接调用存储过程有以下好处：

（1）减少网络通信量。调用一个行数不多的存储过程与直接调用 SQL 语句的网络通信量可能不会有很大的差别，可是如果存储过程包含上百行 SQL 语句，那么其性能绝对比一条一条地调用 SQL 语句要高得多。

（2）执行速度更快。有两个原因：首先，在存储过程创建的时候，数据库已经对其进行了一次解析和优化。其次，存储过程一旦执行，在内存中就会保留一份这个存储过程，这样下



次再执行同样的存储过程时，可以从内存中直接调用。

(3) 更强的适应性。由于存储过程对数据库的访问是通过存储过程接口来进行的，因此数据库开发人员可以在不改动存储过程接口的情况下对数据库进行任何改动，而这些改动不会对应用程序造成影响。

(4) 布式工作。应用程序和数据库的编码工作可以分别独立进行，而不会相互压制。

10.1.2 创建存储过程

创建存储过程，需要使用 CREATE PROCEDURE 语句，基本语法格式如下：

```
CREATE [OR REPLACE] PROCEDURE [schema.] procedure_name
    [parameter_name [[IN]datatype[[:=DEFAULT}expression]]
    {IS|AS}
BODY:
```

CREATE PROCEDURE 为用来创建存储函数的关键字；OR REPLACE 表示如果指定的过程已经存在，则覆盖同名的存储过程；schema 表示该存储过程的所属机构；procedure_name 为存储过程的名称；parameter_name 为存储过程的参数名称；[IN]datatype[[:=DEFAULT}expression]设置传入参数的数据类型和默认值；{IS|AS}表示存储过程的连接词；BODY:表示函数体，是存储过程的具体操作部分，可以用 BEGIN...END 来表示 SQL 代码的开始和结束。

编写存储过程并不是件简单的事情，可能存储过程中需要复杂的 SQL 语句，并且要有创建存储过程的权限；但是使用存储过程将简化操作，减少冗余的操作步骤，同时，还可以减少操作过程中的失误，提高效率，因此存储过程是非常有用的，而且应该尽可能地学会使用。

【例 10.1】创建一个简单的存储过程。输入代码如下：

```
CREATE PROCEDURE HELLO
AS
BEGIN
    dbms_output.put_line('您好，这是一个简单的存储过程');
END;
```

上述代码中，此存储过程名为 HELLO，使用 CREATE PROCEDURE HELLO 语句定义。此存储过程没有参数。BEGIN 和 END 语句用来限定存储过程体，过程本身仅是输出一行字符串。

在 Oracle SQL Developer 中运行上面的代码，执行结果如下：

```
PROCEDURE HELLO 已编译
```




10.2 调用存储过程

存储过程已经定义好了，接下来需要知道如何调用这些存储过程。

直接调用存储过程的方法如下：

```
execute procedure_name;
```

也可以缩写成如下：

```
exec procedure_name;
```

其中，`procedure_name` 为存储过程的名称。

【例 10.2】调用存储过程 HELLO。

在 Oracle SQL Developer 中调用存储过程，如果想让 `DBMS_OUTPUT.PUT_LINE` 成功输出，需要把 `SERVEROUTPUT` 选项设置为 ON 状态。默认情况下，它是 OFF 状态。

可以使用以下语句查看 `SERVEROUTPUT` 选项的状态：

```
SHOW SERVEROUTPUT
```

如果是 OFF 状态，则显示以下信息：

```
SERVEROUTPUT OFF
```

设置为 ON 状态的语句如下：

```
SET SERVEROUTPUT ON
```

在 Oracle SQL Developer 中运行下面的代码调用存储过程 HELLO：

```
exec HELLO;
```

运行结果如下：

```
您好，这是一个简单的存储过程
```

另外，也可以在 `BEGIN...END` 中直接调用存储过程，调用语法如下：

```
BEGIN
    procedure_name;
END;
```

【例 10.3】调用存储过程 HELLO。

```
BEGIN
    HELLO;
```



```
END;
```

运行结果如下：

您好，这是一个简单的存储过程

10.3 查看存储过程

Oracle 存储了存储过程的状态信息，用户可以查看已经存在的存储过程，还可以到视图 `USER_SOURCE` 中查看。

【例 10.4】查看存储过程 HELLO。

查看存储过程的 SQL 语句如下：

```
SELECT * FROM USER_SOURCE WHERE NAME='HELLO' ORDER BY LINE;
```

在 Oracle SQL Developer 中运行上面的代码，结果如下：

NAME	TYPE	LINE	TEXT
HELLO	PROCEDURE	1	CREATE PROCEDURE HELLO
HELLO	PROCEDURE	2	AS
HELLO	PROCEDURE	3	BEGIN
HELLO	PROCEDURE	4	DBMS_OUTPUT.PUT_LINE('您好，这是一个简单的存储过程');
HELLO	PROCEDURE	5	END;

从结果可以看出，每条记录中的 `TEXT` 字段都存储了语句脚本，这些脚本综合起来就是存储过程 HELLO 的内容。

提示

在查看存储过程中，需要把存储过程的名称大写，如果小写，则无法查询到任何内容。如果想查看所有的存储过程，可以在 `ALL_SOURCE` 视图中查询。

10.4 存储过程的参数

存储过程可以带参数，也可以不带参数。在数据转换时经常使用不带参数的存储过程。

10.4.1 无参数的存储过程

下面通过案例来学习不带参数的存储过程的使用方法和技巧。



Oracle 12c 从零开始学（视频教学版）

【例 10.5】把数据表 fruits 中价格最低的两个水果的名称设置为“打折水果”。
创建存储过程的脚本如下：

```
CREATE PROCEDURE FRUITS_PRC
AS
BEGIN
UPDATE fruits SET f_name='打折水果'
WHERE f_id IN
(
SELECT f_id FROM
(SELECT * FROM fruits ORDER BY f_price ASC)
WHERE ROWNUM<3
);
COMMIT;
END;
```

其中，COMMIT;表示提交更改。

在 Oracle SQL Developer 中运行上面的代码，结果如下：

PROCEDURE FRUITS_PRC 已编译

在 Oracle SQL Developer 中执行存储过程 FRUITS_PRC，语句如下：

```
EXEC FRUITS_PRC;
```

查看数据表 fruits 的记录是否发生变化，SQL 语句如下：

```
SELECT * FROM fruits WHERE ROWNUM<3;

F_ID      F_NAME
-----
a2        打折水果
m2        打折水果
c0        cherry
b5        xxxx
t4        xbababa
a1        apple
t2        grape
l2        lemon
b2        berry
```



```
bs2      melon
o2       coconut
b1       blackberry
t1       banana
bs1      orange
m3       xxtt
m1       mango
```

从结果可以看出，存储过程已经生效。

10.4.2 有参数的存储过程

存储过程可以带有参数，使用参数可以增加存储过程的灵活性，为数据库编程带来很大的便利。存储过程的参数可以是常量、变量和表达式等。一旦为存储过程使用了参数，在执行存储过程时，必须指定对应的参数。

【例 10.6】根据输入的水果类型编码，在数据表 `fruits` 中搜索符合条件的数据，并将数据输出。

创建存储过程的脚本如下：

```
CREATE PROCEDURE FRUITS_PRCC(parm_sid IN NUMBER(6))
AS
    cur_id fruits.f_id%type;           --存放水果的编码
    cur_prtifo fruits%ROWTYPE;        --存放表 fruits 的行记录

BEGIN
    SELECT fruits.f_id INTO cur_id
    FROM fruits
    WHERE s_id = parm_sid;             --根据水果类型编码获取水果的编码
    IF SQL%FOUND THEN
        DBMS_OUTPUT.PUT_LINE(parm_sid||':');
    END IF;
    FOR my_prdinfo_rec IN
    (
        SELECT * FROM fruits WHERE CATEGORY=cur_id)
    LOOP
        DBMS_OUTPUT.PUT_LINE('水果名称'|| my_prdinfo_rec.f_name
        ||'水果价格'|| my_prdinfo_rec.f_price
```




Oracle 12c 从零开始学（视频教学版）

```
||'水果数量'|| my_prdinfo_rec.QUANTITY);  
        END LOOP;  
    EXCEPTION  
        WHEN NO_DATA_FOUND THEN  
            DBMS_OUTPUT.PUT_LINE('没有数据');  
    WHEN TOO_MANY_ROWS THEN  
        DBMS_OUTPUT.PUT_LINE('数据过多');  
END;
```

在 Oracle SQL Developer 中运行上面的代码，结果如下：

```
PROCEDURE FRUITS_PRCC 已编译
```

在 Oracle SQL Developer 中执行存储过程 FRUITS_PRCC，语句如下：

```
EXEC FRUITS_PRCC (106);
```

执行结果如下：

```
106;  
水果名称 mango 水果价格 15.6 水果数量1
```

10.5 修改存储过程

在 Oracle 中，如果要修改存储过程，使用 CREATE OR REPLACE PROCEDURE 语句，也就是覆盖原始的存储过程。

【例 10.7】修改存储过程 HELLO，代码如下：

```
CREATE OR REPLACE PROCEDURE HELLO  
AS  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('这是修改后的存储过程');  
END;
```

在 Oracle SQL Developer 中运行下面的代码，调用存储过程 HELLO：

```
exec HELLO;
```



运行结果如下：

这是修改后的存储过程

10.6 删除存储过程

删除存储过程，可以使用 DROP 语句，其语法结构如下：

```
DROP PROCEDURE [schema.] procedure_name
```

Schema 表示存储过程所属的机构；procedure_name 为要移除的存储过程的名称。

【例 10.8】删除存储过程 HELLO，代码如下：

```
DROP PROCEDURE HELLO;
```

上面语句的作用就是删除存储过程 HELLO。

10.7 查看存储过程的错误

编写的存储过程难免会出现各种错误而导致编译失败，为了减少排查错误的范围，Oracle 提供了查看存储过程错误的方法，语法结构如下：

```
SHOW ERRORS PROCEDURE procedure_name;
```

【例 10.9】创建一个有错误的存储过程，然后查看错误信息。

创建一个有错误的存储过程，运行代码如下：

```
CREATE OR REPLACE PROCEDURE HELLO  
AS  
BEGIN  
  DBMM_OUTPUT.PUT_LINE('这是有错误的存储过程');  
END;
```

执行结果如下：

警告：创建的过程带有编译错误

查看错误的具体细节，代码如下：

```
SHOW ERRORS PROCEDURE HELLO;
```




结果如下：

```
Errors: check compiler log
4/1          PLS-00201: 必须声明标识符 'DBMM_OUTPUT.PUT_LINE'
4/1          PL/SQL: Statement ignored
```

从错误提示可知，错误是由第 4 行引起，正确的写法如下：

```
DBMS_OUTPUT.PUT_LINE('这是有错误的存储过程');
```

10.8 综合案例——综合运用存储过程

通过这一章的学习，读者应该掌握了如何运用存储过程以及 Oracle 的控制语句。所有的存储过程都存储在服务器上，只要调用就可以在服务器上执行。

1. 案例目的

通过实例掌握存储过程的创建和使用。

2. 案例操作过程

步骤 01 创建一个名称为 sch 的数据表，表结构如表 10-1 所示，将表 10-2 中的数据插入到 sch 表中。

表 10-1 sch 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
id	NUMBER(10)	是	否	是	是	否
name	VARCHAR2 (50)	否	否	是	否	否
glass	VARCHAR2(50)	否	否	是	否	否

表 10-2 sch 表内容

id	name	glass
1	xiaoming	glass 1
2	xiaojun	glass 2

创建一个 sch 表，并且向 sch 表中插入数据，代码如下：

```
CREATE TABLE sch(id NUMBER(10), name VARCHAR2(50),glass VARCHAR2(50));
INSERT INTO sch VALUES(1,'xiaoming','glass 1');
INSERT INTO sch VALUES(2,'xiaojun','glass 2');
```

通过 DESC 命令查看创建的表格，结果如下：

```
SQL> DESC sch;
名称 空值 类型
-----
ID          NUMBER(10)
NAME        VARCHAR2(50)
GLASS       VARCHAR2(50)
```

通过 SELECT * FROM sch 来查看插入表格的内容，结果如下：

```
ID  NAME  GLASS
-----
1   xiaoming  glass 1
2   xiaojun  glass 2
```

步骤 02 创建一个存储过程用来统计表 sch 中的记录数和 sch 表中 id 的和。

创建一个可以统计表格内记录条数的存储过程，存储过程名为 COUNT_SCH，代码如下：

```
CREATE OR REPLACE PROCEDURE COUNT_SCH
AS
cur_count number(6);
cur_sum number(6);
BEGIN
    SELECT COUNT(*) INTO cur_count
    FROM sch;
    SELECT SUM(id) INTO cur_sum
    FROM sch;

    IF SQL%FOUND THEN
        DBMS_OUTPUT.PUT_LINE('记录总数: '||cur_count);
        DBMS_OUTPUT.PUT_LINE('id 总和: '||cur_sum);
    END IF;
END;
```

在 Oracle SQL Developer 中运行上面的代码，结果如下：

```
PROCEDURE COUNT_SCH 已编译
```


在 Oracle SQL Developer 中执行存储过程 COUNT_SCH，语句如下：

```
EXEC COUNT_SCH;
```

执行结果如下：

记录总数：2

id 总和：3

10.9 疑难解惑

疑问 1：存储过程中的代码可以改变吗？

目前，Oracle 还不提供对已存在的存储过程代码的修改，如果必须要修改存储过程，可以使用 CREATE OR REPLACE PROCEDURE 语句覆盖掉同名的存储过程。

疑问 2：删除存储过程需要注意什么问题？

存储过程之间可以相互调用，如果删除被调用的存储过程，那么重新编译时调用者会出现错误，所以在执行删除操作时，最好要分清各个存储过程之间的关系。

10.10 经典习题

- (1) 创建一个名为 WORD 的存储过程，功能读者可以自定。
- (2) 调用存储过程 WORD。
- (3) 修改存储过程 WORD。

第 11 章

◀ Oracle 触发器 ▶

Oracle 的触发器和存储过程一样，都是嵌入到 Oracle 的一段程序。触发器是由事件来触发某个操作，这些事件包括 INSERT、UPDATE 和 DELETE 语句。如果定义了触发程序，当数据库执行这些语句的时候就会激发触发器执行相应的操作，触发程序是与表有关的命名数据库对象，当表上出现特定事件时，将激活该对象。本章通过实例来介绍触发器的含义、如何创建触发器、查看触发器、触发器的使用方法以及如何删除触发器。

- 了解什么是触发器
- 掌握创建触发器的方法
- 掌握查看触发器的方法
- 掌握触发器的使用技巧
- 掌握删除触发器的方法
- 熟练掌握综合案例中使用触发器的方法和技巧

11.1 创建触发器

学习了存储过程后，再学习触发器会比较容易，它们之间有很多相似的地方。本节将介绍如何创建触发器。

11.1.1 什么是触发器

触发器 (trigger) 是个特殊的存储过程，不同的是，执行存储过程要使用 EXEC 语句来调用，而触发器的执行不需要使用 EXEC 语句来调用，也不需要手工启动，只要当一个预定义的事件发生的时候，就会被 Oracle 自动调用。比如，当对 fruits 表进行操作 (INSERT、DELETE 或 UPDATE) 时就会激活它执行。

触发器可以查询其他表，而且可以包含复杂的 SQL 语句。它主要用于满足复杂的业务规则或要求。例如，可以根据客户当前的账户状态，控制是否允许插入新订单。

11.1.2 创建只有一个执行语句的触发器

创建只有一个执行语句的触发器的语法如下：

```
CREATE TRIGGER TRIGGER_NAME trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_stmt
```

其中，TRIGGER_NAME 标识触发器名称，用户自行指定；trigger_time 标识触发时机，可以指定为 before 或 after；trigger_event 标识触发事件，包括 INSERT、UPDATE 和 DELETE；tbl_name 标识建立触发器的表名，即在哪张表上建立触发器；trigger_stmt 是触发器程序体。触发器程序可以使用 BEGIN 和 END 作为开始和结束，中间包含多条语句。这一小节来学习单执行语句的触发器。

【例 11.1】创建一个单执行语句的触发器。

首先创建一个 account 表，表中有两个字段，分别为：acct_num 字段（定义为整数类型），amount 字段（定义成小数类型），代码如下：

```
CREATE TABLE account (acct_num NUMBER(6), amount NUMBER(10,2));
```

其次创建一个名为 INS_SUM 的触发器，触发的条件是向数据表 account 插入数据之前，输出提示信息，代码如下：

```
CREATE TRIGGER INS_SUM
BEFORE INSERT
ON account
BEGIN
IF INSERT THEN
    DBMS_OUTPUT.PUT_LINE('下面将开始插入数据');
END IF;
END ;
```

检验触发器是否生效，向表中插入数据：

```
INSERT INTO account VALUES (1,10.22);
```

执行结果如下：

```
下面将开始插入数据
1 行已插入。
```

从结果可以看出，在插入数据前，启动了触发器。

11.1.3 创建有多个执行语句的触发器

创建有多个执行语句的触发器的语法如下：

```
CREATE TRIGGER TRIGGER_NAME trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_stmt
```

其中, TRIGGER_NAME 标识触发器的名称, 用户自行指定; trigger_time 标识触发时机, 可以指定为 before 或 after; trigger_event 标识触发事件, 包括 INSERT、UPDATE 和 DELETE; tbl_name 标识建立触发器的表名, 即在哪张表上建立触发器; trigger_stmt 是触发器程序体。触发器程序可以使用 BEGIN 和 END 作为开始和结束, 中间包含多条语句。这一小节来学习执行多个语句的触发器。

【例 11.2】创建一个包含多个执行语句的触发器。

创建 4 张表, 代码如下:

```
CREATE TABLE test1(a1 NUMBER(6));
CREATE TABLE test2(a2 NUMBER(6));
CREATE TABLE test3(a3 NUMBER(6) GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY);
CREATE TABLE test4(
    a4 NUMBER(6) GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    b4 NUMBER(6) DEFAULT 0
);
```

创建一个包含多个执行语句的触发器, 代码如下:

```
CREATE TRIGGER TESTREF
BEFORE INSERT
ON test1
FOR EACH ROW
BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
END
```

上面的代码是创建了一个名为 TESTREF 的触发器, 这个触发器的触发条件是在向表 test1 插入数据前执行触发器的语句, 具体执行的代码如下:

```
SQL> INSERT INTO test1 VALUES (1);
SQL> INSERT INTO test1 VALUES (3);
SQL> INSERT INTO test1 VALUES (1);
SQL> INSERT INTO test1 VALUES (7);
```




```
SQL> INSERT INTO test1 VALUES (1);
SQL> INSERT INTO test1 VALUES (8);
SQL> INSERT INTO test1 VALUES (4);
SQL> INSERT INTO test1 VALUES (4);
```

那么 4 个表中的数据如下：

```
SQL> SELECT * FROM test1;
```

A1

1

3

1

7

1

8

4

4

```
SQL> SELECT * FROM test2;
```

A2

1

3

1

7

1

8

4

4

```
SQL> SELECT * FROM test3;
```

A3



```
-----  
2  
5  
6  
9  
10
```

```
SQL> SELECT * FROM test4;
```

```
A4  B4
```

```
-----  
1  3  
2  0  
3  1  
4  2  
5  0  
6  0  
7  1  
8  1  
9  0  
10 0
```

执行结果显示，在向表 test1 插入记录的时候，test2、test3、test4 都发生了变化。从这个例子看 INSERT 触发了触发器，向 test2 中插入了 test1 中的值，删除了 test3 中相同的内容，同时更新了 test4 中的 b4，即与插入的值相同的个数。

11.2 查看触发器

查看触发器是指查看数据库中已存在的触发器的定义、状态和语法信息等。用户可以通过命令来查看已经创建的触发器。

11.2.1 查看触发器的名称

用户可以查看已经存在的触发器的名称。



【例 11.3】查看触发器名称的命令如下：

```
SELECT OBJECT_NAME FROM USER_OBJECTS
WHERE OBJECT_TYOE='TRIGGER';
```

结果如下：

```
OBJECT_NAME
-----
INS_SUM
TESTREF
```

11.2.2 查看触发器的内容信息

有了触发器的名称，就可以查看触发器的具体内容了。

【例 11.4】查看触发器 INS_SUM 的内容信息，命令如下：

```
SELECT * FROM USER_SOURCE WHERE NAME= ' INS_SUM' ORDER BY LINE;
```

结果如下：

NAME	TYPE	LINE	TEXT
INS_SUM	TRIGGER	1	CREATE TRIGGER INS_SUM
INS_SUM	TRIGGER	2	BEFORE INSERT
INS_SUM	TRIGGER	3	ON account
INS_SUM	TRIGGER	4	BEGIN
INS_SUM	TRIGGER	5	IF INSERT THEN
INS_SUM	TRIGGER	6	DBMS_OUTPUT.PUT_LINE('下面将开始插入数据');
INS_SUM	TRIGGER	7	END IF;
INS_SUM	TRIGGER	8	END ;

11.3 触发器的使用

触发程序是与表有关的命名数据库对象，当表上出现特定事件时，将激活该对象。在某些触发程序的用法中，可用于检查插入到表中的值，或对更新涉及的值进行计算。

触发程序与表相关，当对表执行 INSERT、DELETE 或 UPDATE 语句时，将激活触发程序。可以将触发程序设置为在执行语句之前或之后激活。例如，可以在从表中删除每一行之前，或在更新每一行之后激活触发程序。



【例 11.5】创建一个在 account 表插入记录之后，更新 myevent 数据表的触发器。

数据表 myevent 定义如下：

```
CREATE TABLE myevent
(
  id NUMBER(11) DEFAULT NULL,
  evt_name VARCHAR2(20) DEFAULT NULL
);
```

创建一个 TRIG_INSERT 触发器，在向表 account 插入数据之后会向表 myevent 插入一组数据，代码执行如下：

```
CREATE TRIGGER TRIG_INSERT
AFTER INSERT
ON account
BEGIN
  IF INSERT THEN
    INSERT INTO myevent VALUES (2,'after insert');
  END IF;
END ;
```

检验触发器的效果，代码如下，

```
SQL> INSERT INTO account VALUES (1,1.00);
SQL> INSERT INTO account VALUES (2,2.00);

SQL> SELECT * FROM myevent;
```

```
  ID  EVT_NAME
-----
  2   after insert
  2   after insert
```

从执行的结果来看，是创建了一个名称为 TRIG_INSERT 的触发器，它是在向表 account 插入记录之后进行触发，执行的操作是向表 myevent 插入一条记录。



11.4 修改触发器

在 Oracle 中，如果要修改触发器，使用 CREATE OR REPLACE TRIGGER 语句，也就是覆盖原始的触发器。

【例 11.6】修改触发器 INS_SUM，代码如下：

```
CREATE OR REPLACE TRIGGER INS_SUM
BEFORE INSERT
ON account
BEGIN
    IF INSERT THEN
        DBMS_OUTPUT.PUT_LINE('这是修改后的触发器');
    END IF;
END ;
```

检验触发器是否被修改，向表中插入数据：

```
INSERT INTO account VALUES (2,10.22);
```

执行结果如下：

```
这是修改后的触发器
1 行已插入。
```

从结果可以看出，触发器被成功修改。

11.5 删除触发器

使用 DROP TRIGGER 语句可以删除 Oracle 中已经定义的触发器，删除触发器语句的基本语法格式如下：

```
DROP TRIGGER [schema.]TRIGGER_NAME
```

其中，schema 表示该触发器的所属机构，是可选的；TRIGGER_NAME 是要删除的触发器的名称。

【例 11.7】删除一个触发器，代码如下：

```
DROP TRIGGER INS_SUM;
```



11.6 综合案例——使用触发器

本章介绍了 Oracle 数据库中触发器的定义和作用、创建触发器、查看触发器、使用触发器和删除触发器等内容。创建触发器和使用触发器是本章的重点内容。在创建触发器的时候，一定要弄清楚触发器的结构；在使用触发器的时候，要清楚触发器触发的时间（BEFORE 或 AFTER）和触发的条件（INSERT、DELETE 或 UPDATE）。在创建触发器后，要清楚怎么修改触发器。

1. 案例目的

掌握触发器的创建和调用方法。

下面是创建触发器的实例，每更新一次 persons 表的 num 字段后，都要更新 sales 表对应的 sum 字段。其中，persons 表结构如表 11-1 所示，sales 表结构如表 11-2 所示，persons 表内容如表 11-3 所示，按照操作过程完成操作。

表 11-1 persons 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
name	VARCHAR2 (40)	否	否	是	否	否
num	NUMBER (11)	否	否	是	否	否

表 11-2 sales 表结构

字段名	数据类型	主键	外键	非空	唯一	自增
name	VARCHAR2 (40)	否	否	是	否	否
sum	NUMBER (11)	否	否	是	否	否

表 11-3 persons 表内容

name	num
Xiaoxiao	20
xiaohua	69

2. 案例操作过程

步骤 01 创建一个业务统计表 persons。

创建一个业务统计表 persons，代码如下：

```
CREATE TABLE persons (name VARCHAR2(40), num NUMBER(11));
```

步骤 02 创建一个销售额表 sales。

创建一个销售额表 sales，代码如下：



```
CREATE TABLE sales (name VARCHAR2(40), sum NUMBER(11));
```

步骤 03 创建一个触发器。

创建一个触发器，在插入 persons 表的 num 字段后，更新 sales 表的 sum 字段，代码如下：

```
CREATE TRIGGER NUM_SUM
AFTER INSERT
ON persons
BEGIN
    IF INSERT THEN
        INSERT INTO sales VALUES (NEW.name, 7*NEW.num);
    END IF;
END ;
```

步骤 04 向 persons 表中插入记录，代码如下：

```
INSERT INTO persons VALUES ('xiaoxiao', 20);
INSERT INTO persons VALUES ('xiaohua', 69);
```

步骤 05 查询 persons 表中的记录，代码如下：

```
SQL> SELECT * FROM persons;
```

NAME	NUM
------	-----

xiaoxiao	20
----------	----

xiaohua	69
---------	----

步骤 06 查询 sales 表中的记录，代码如下：

```
SQL> SELECT * FROM sales;
```

NAME	SUM
------	-----

xiaoxiao	140
----------	-----

xiaohua	483
---------	-----

从执行的结果来看，在 persons 表插入记录之后，NUM_SUM 触发器计算插入到 persons 表中的数据，并将结果插入到 sales 表中相应的位置。



11.7 疑难解惑

疑问 1：创建触发器时需特别注意什么问题？

在使用触发器的时候需要注意，对于相同的表，相同的事件只能创建一个触发器，比如对表 account 创建了一个 BEFORE INSERT 触发器，那么如果对表 account 再次创建一个 BEFORE INSERT 触发器，Oracle 将会报错。此时，只可以在表 account 上创建 AFTER INSERT 或者 BEFORE UPDATE 类型的触发器。灵活地运用触发器将为操作省去很多麻烦。

疑问 2：为什么要及时删除不用的触发器？

触发器定义之后，每次执行触发事件，都会激活触发器并执行触发器中的语句。如果需求发生变化，而触发器没有进行相应的改变或者删除，则触发器仍然会执行旧的语句，从而影响新数据的完整性。因此，要将不再使用的触发器及时删除。

11.8 经典习题

- (1) 创建 INSERT 事件的触发器。
- (2) 创建 UPDATE 事件的触发器。
- (3) 创建 DELETE 事件的触发器。
- (4) 查看触发器。
- (5) 删除触发器。



第 12 章

◀ 管理表空间 ▶

在 Oracle 中，创建数据库时需要同时指定数据库建立的表空间，所以表空间是非常重要的知识。本章主要介绍表空间的基础知识和管理表空间的方法等内容。通过本章的学习，读者可以在创建数据库时熟练地指定表空间和大文件表空间。

- 了解什么是表空间
- 掌握查看表空间的方法
- 掌握管理表空间的方法
- 掌握管理临时表空间的方法
- 掌握管理数据文件的方法

12.1 什么是表空间

表空间是 Oracle 数据库的必备知识。本节主要讲述表空间的基本知识。

表空间是数据库的逻辑划分，Oracle 数据库被划分成多个表空间的逻辑区域，这样就形成了 Oracle 数据库的逻辑结构。一个表空间只能属于一个数据库。所有的数据库对象都存放在指定的表空间中。但主要存放的是表，所以称作表空间。一个 Oracle 数据库能够有一个或多个表空间，而一个表空间则对应着一个或多个物理的数据库文件。表空间是 Oracle 数据库恢复的最小单位，容纳着许多数据库实体，如表、视图、索引、聚簇、回退段和临时段等。

Oracle 数据库中至少存在一个表空间，即 SYSTEM 表空间。每个 Oracle 数据库均有 SYSTEM 表空间，这是数据库创建时自动创建的。SYSTEM 表空间必须总要保持联机，因为其包含着数据库运行所要求的基本信息，包括关于整个数据库的数据字典、联机求助机制、所有回退段、临时段和自举段、所有的用户数据库实体、其他 Oracle 软件产品要求的表。

一个小型应用的 Oracle 数据库通常仅包括 SYSTEM 表空间，然而一个稍大型应用的 Oracle 数据库采用多个表空间会给数据库的使用带来更大的方便。

Oracle 表空间的作用是能帮助 DBA 用户完成以下工作：

- (1) 决定数据库实体的空间分配；
- (2) 设置数据库用户的空间份额；



- (3) 控制数据库部分数据的可用性;
- (4) 分布数据于不同的设备之间以改善性能;
- (5) 备份和恢复数据。

用户创建其数据库实体时必须于给定的表空间中具有相应的权力, 所以对一个用户来说, 如要操纵一个 Oracle 数据库中的数据, 应该:

- (1) 被授予关于一个或多个表空间中的 RESOURCE 特权;
- (2) 被指定默认表空间;
- (3) 被分配指定表空间的存储空间使用份额;
- (4) 被指定默认临时段表空间, 建立不同的表空间, 设置最大的存储容量。

12.2 查看表空间

在 Oracle 12c 中, 默认的表空间有 5 个, 分别为 SYSTEM、SYSAUX、UNDOTBS1、TEMP 和 USERS。

【例 12.1】查询当前登录用户默认的表空间的名称, SQL 语句如下:

```
SELECT TABLESPACE_NAME FROM DBA_TABLESPACES;
```

查询结果如下:

```
TABSPACE_NAME
```

```
-----
```

```
SYSTEM
```

```
SYSAUX
```

```
UNDOTBS1
```

```
TEMP
```

```
USERS
```

从结果可以看出, 默认情况下有 5 个表空间。各个表空间的含义如下:

- (1) SYSTEM 表空间: 用来存储 SYS 用户的表、视图和存储过程等数据库对象。
- (2) SYSAUX 表空间: 用于安装 Oracle 12c 数据库使用的实例数据库。
- (3) UNDOTBS1 表空间: 用于存储撤销信息。
- (4) TEMP 表空间: 用户存储 SQL 语句处理的表和索引的信息。
- (5) USERS 表空间: 存储数据库用户创建的数据库对象。

如果要查看某个用户的默认表空间, 可以通过 DBA_USERS 数据字典进行查询。

【例 12.2】查询 SYS、SYSDG、SYSBACKUP、SYSTEM 和 SYSKM 用户的默认表空间，SQL 语句如下：

```
SELECT DEFAULT_TABLESPACE, USERNAME FROM DBA_USERS WHERE USERNAME LIKE 'SYS%';
```

查询结果如下：

DEFAULT_TABLESPACE	USERNAME
USERS	SYSDG
USERS	SYSKM
USERS	SYSBACKUP
SYSTEM	SYSTEM
SYSTEM	SYS

从结果可以看出，SYSDG、SYSBACKUP 和 SYSKM 用户的默认表空间是 USERS，SYS 和 SYSTEM 用户的默认表空间是 SYSTEM。

如果想要查看表空间的使用情况，可以使用数据字典 DBA_FREE_SPACE。

【例 12.3】查询 SYSTEM 默认表空间的使用情况，SQL 语句如下：

```
SELECT * FROM DBA_FREE_SPACE WHERE TABLESPACE_NAME='SYSTEM';
```

结果如下：

TABLESPACE_NAME	FILE_ID	BLOCK_ID	BYTES	BLOCKS	RELATIVE_FNO
SYSTEM	1	103216	3801088	464	1

12.3 管理表空间

在 Oracle 数据库中，用户可以对表空间进行创建、修改和删除等操作。

12.3.1 创建表空间

使用 CREATE TABLESPACE 语句可以创建表空间，语法规则如下：

```
CREATE TABLESPACE tablespace_name
DATAFILE filename SIZE size
[AUTOEXTEND[ON/OFF]]NEXT size
```

```
[MAXSIZE size]
[PERMANENT|TEMPORARY]
[EXTENT MANAGEMENT
[DICTIONARY|LOCAL
[AUTOALLOCATE|UNIFORM.[SIZE integer[K|M]]]]]
```

上述代码的具体含义如下：

- `tablespace_name`: 创建表空间的名称。
- `DATAFILE filename SIZE size`: 指定在表空间中存放数据文件的文件名和数据库文件的大小。
- `[AUTOEXTEND[ON/OFF]]NEXT size`: 指定数据文件的扩展方式, ON 代表自动扩展, OFF 为为非自动扩展, NEXT 后指定自动扩展的大小。
- `[MAXSIZE size]`: 指定数据文件为自动扩展方式时的最大值。
- `[PERMANENT|TEMPORARY]`: 指定表空间的类型, PERMANENT 表示永久表空间, TEMPORARY 表示临时性表空间。如果不指定表空间的类型, 默认为永久性表空间。
- `EXTENT MANAGEMENT DICTIONARY|LOCAL`: 指定表空间的管理方式, DICTIONARY 是指字典管理方式, LOCAL 是指本地管理方式。默认情况下的管理方式为本地管理方式。

【例 12.4】创建一个表空间, 名称为 MYTEM, 大小为 30MB, SQL 语句如下:

```
CREATE TABLESPACE MYTEM
DATAFILE 'MYTEM.DBF' SIZE 30M;
```

结果如下:

tablespace MYTEM 已创建。

其中, MYTEM.DBF 为表空间的数据文件。

【例 12.5】创建一个表空间, 名称为 MYTEMM, 大小为 20MB, 可以自动扩展, 每次扩展 256KB, 最大只允许扩展到 2048MB, SQL 语句如下:

```
CREATE TABLESPACE MYTEMM
DATAFILE 'MYTEMM.DBF' SIZE 20M
AUTOEXTEND ON NEXT 256KB
MAXSIZE 2048M;
```

结果如下:

tablespace MYTEMM 已创建。

12.3.2 设置表空间的可用状态

表空间的可用状态为两种：联机状态和脱机状态。如果是联机状态，此时用户可以操作表空间；如果是脱机状态，此时表空间是不可用的。

设置表空间的可用状态的语法格式如下：

```
ALTER TABLESPACE tablespace {ONLINE|OFFLINE[NORMAL|TEMPORARY|IMMEDIATE]}
```

其中，ONLINE 表示设置表空间为联机状态；OFFLINE 为脱机状态，包括 NORMAL 为正常状态、TEMPORARY 为临时状态、IMMEDIATE 为立即状态。

【例 12.6】把表空间 MYTEMM 设置为脱机状态，脱机状态为临时状态，SQL 语句如下：

```
ALTER TABLESPACE MYTEMM OFFLINE TEMPORARY;
```

结果如下：

```
tablespace MYTEMM 已变更。
```

如果想恢复表空间 MYTEMM 为联机状态，可用以下语句：

```
ALTER TABLESPACE MYTEMM ONLINE;
```

12.3.3 设置表空间的读写状态

根据需要，用户可以把表空间设置成只读或者可读写状态。具体的语法格式如下：

```
ALTER TABLESPACE tablespace READ {ONLY|WRITE} ;
```

其中，ONLY 为只读状态；WRITE 为可以读写状态。

【例 12.7】把表空间 MYTEMM 设置为只读状态，SQL 语句如下：

```
ALTER TABLESPACE MYTEMM READ ONLY;
```

结果如下：

```
tablespace MYTEMM 已变更。
```

【例 12.8】把表空间 MYTEMM 设置为可读写状态，SQL 语句如下：

```
ALTER TABLESPACE MYTEMM READ WRITE;
```

结果如下：

```
tablespace MYTEMM 已变更。
```

在设置表空间为只读状态之前，需要保证表空间为联机状态。

12.3.4 重命名表空间

对于已经存在的表空间，可以根据需要更改名称。语法格式如下：

```
ALTER TABLESPACE oldname RENAME TO newname;
```

【例 12.9】把表空间 MYTEMM 的名称更改为 MYTEMNEW，SQL 语句如下：

```
ALTER TABLESPACE MYTEMM RENAME TO MYTEMNEW;
```

结果如下：

```
tablespace MYTEMM 已变更。
```

并不是所有的表空间都可以命名，系统自动创建的不可更名，例如 SYSTEM 和 SYSAUX 等；另外，表空间必须是联机状态才可以重命名。

12.3.5 删除表空间

删除表空间的方式有两种，包括使用本地管理方式和使用数据字典的方式。相比而言，使用本地管理方式删除表空间的速度更快些。所以在删除表空间前，可以先把表空间的管理方式修改为本地管理，然后再删除表空间。

删除表空间的语法格式如下：

```
DROP TABLESPACE tablespace_name [INCLUDING CONTENTS] [CASCADE CONSTRAINTS];
```

其中，[INCLUDING CONTENTS]表示在删除表空间时把表空间文件也删除；[CASCADE CONSTRAINTS]表示在删除表空间时把表空间中的完整性也删除。

【例 12.10】删除表空间 MYTEM，SQL 语句如下：

```
DROP TABLESPACE MYTEM INCLUDING CONTENTS;
```

结果如下：

```
tablespace MYTEM 已删除。
```

12.3.6 建立大文件表空间

创建大文件表空间和普通表空间的语法格式非常类似，定义大文件表空间的语法格式如下：

```
CREATE BIGFILE TABLESPACE tablespace_name  
DATAFILE filename SIZE size
```

【例 12.11】建立大文件表空间，名称为 MYBG，SQL 语句如下：

```
CREATE BIGFILE TABLESPACE MYBG
```



```
DATAFILE mybg.dbf SIZE 3G;
```

结果如下：

```
tablespace MYBG 已创建.
```

创建了 3GB 的数据文件 mybg.dbf。

12.4 管理临时表空间

临时表空间也是表空间的一种，主要用来存放临时的数据信息。本章节主要讲述临时表空间的一些操作。

12.4.1 创建临时表空间

在数据库中执行存储操作时，如果内存不够可以写入临时表空间，当执行完对数据库的操作后，该空间的内容自动清空。

创建临时表空间的语法格式如下：

```
CREATE TEMPORARY TABLESPACE tablespace_name  
TEMPFILE filename SIZE size
```

【例 12.12】创建一个临时表空间，名称为 MYTT，大小为 30MB，SQL 语句如下：

```
CREATE TEMPORARY TABLESPACE MYTT  
TEMPFILE 'MYTT.DBF' SIZE 30M;
```

结果如下：

```
tablespace MYTT 已创建.
```

对于已经存在的表空间，可以修改为临时表空间。

【例 12.13】把表空间 MYTEM 修改为临时表空间，SQL 语句如下：

```
ALTER DEFAULT TEMPFILE TABLESPACE MYNTT
```

结果如下：

```
tablespace MYTEM 已更改.
```

12.4.2 查看临时表空间

使用数据字典 DBA_TEMP_FILES 可以查看临时表空间。

【例 12.14】查询临时表空间的名称，SQL 语句如下：

```
SELECT TABLESPACE_NAME FROM DBA_TEMP_FILES;
```

查询结果如下：

```
TABLESPACE_NAME
```

```
-----
```

```
MYTT
```

```
MYNTT
```

12.4.3 创建临时表空间组

临时表空间组是由多个表空间组成的，每一个临时表空间组至少要包含一个临时表空间，而且临时表空间组的名称不能和其他表空间重名。

创建临时表空间组的语法格式如下：

```
CREATE TEMPORARY TABLESPACE tablespace_name
TEMPFILE filename SIZE size TABLESPACE GROUP group_name;
```

【例 12.15】创建一个临时表空间组，名称为 testgroup，大小为 20MB，SQL 语句如下：

```
CREATE TEMPORARY TABLESPACE MYNTT
TEMPFILE 'MYNTT.DBF' SIZE 20M TABLESPACE GROUP testgroup;
```

对于已经存在的临时表空间，可以将其移动到指定的临时表空间组中。

【例 12.16】将临时表空间 MYTT 移到临时表空间组 testgroup 中，SQL 语句如下：

```
ALTER TABLESPACE MYTT GROUP testgroup;
```

12.4.4 查看临时表空间组

通过数据字典 DBA_TABLESPACE_GROUPS，可以查看临时表空间组信息。

【例 12.17】查看临时表空间组信息，SQL 语句如下：

```
SELECT * FROM DBA_TABLESPACE_GROUPS;
```

查询结果如下：

```
GROUP_NAME    TABLESPACE_NAME
```

```
-----
```

```
TESTGROUP     MYTT
```

```
TESTGROUP     MYNTT
```


12.4.5 删除临时表空间组

在删除临时表空间组时，需要把临时表空间组中的临时表空间一起删除。具体删除临时表空间组的语法格式如下：

```
DROP TABLESPACE tablespace_name INCLUDING CONTENTS AND DATAFILES;
```

【例 12.18】删除临时表空间组 testgroup，SQL 语句如下：

```
DROP TABLESPACE TESTGROUP INCLUDING CONTENTS AND DATAFILES;
```

在删除临时表空间组时，不能删除默认的临时表空间。

12.5 管理数据文件

在创建和管理表空间时，都会用到数据文件。本节主要讲述数据文件的一些操作。

12.5.1 移动数据文件

在 Oracle 数据库中，创建表空间时，数据文件也同时被创建了。根据实际工作的需要，可以把当前表空间中的数据文件移动到其他表空间中。

移动数据文件的基本步骤如下：

(1) 把要存放数据文件所用的表空间设置成脱机状态。语句如下：

```
ALTER TABLESPACE MYTEMM OFFLINE;
```

(2) 可以手动把要移动的文件移动到其他的表空间中。

(3) 更改数据文件的名称。语句如下：

```
ALTER TABLESPACE MYTEMM RENAME oldfilename TO newfilename;
```

(4) 把该表空间设置成联机状态。语句如下：

```
ALTER TABLESPACE MYTEMM ONLINE;
```

12.5.2 删除数据文件

对应不再使用的数据文件，可以进行删除操作。如果数据文件处于以下 3 种情况，不可以被删除。

- (1) 数据文件或者数据文件所在的表空间处于只读状态。
- (2) 数据文件中存在数据。
- (3) 数据文件是表空间中唯一一个或第一个数据文件。

12.6 疑难解惑

疑问 1：临时表空间组删除后能恢复吗？

临时表空间组删除后不能恢复，所以在执行删除操作时必须慎重。在删除临时表空间组后，临时表空间组的文件并没有删除，如果要删除临时表空间组的临时表空间，需要先把临时表空间组中的临时表空间移除。

疑问 2：创建表空间时用什么方式？

使用本地管理的方式可以减少数据字典的竞争现象，并且也不需要表空间进行回收，因此，在 Oracle 中最好使用本地管理的方式创建表空间。

12.7 经典习题

- (1) 简述什么是表空间。
- (2) 创建一个表空间并进行修改操作。
- (3) 创建一个临时表空间。
- (4) 创建一个临时表空间组。
- (5) 移动数据文件到指定的表空间中。
- (6) 删除数据文件。

第 13 章

◀ 事务与锁 ▶

Oracle 中提供了多种数据完整性的保证机制，如约束、触发器、事务和锁管理等。事务管理主要是为了保证一批相关数据库中数据的操作能全部被完成，从而保证数据的完整性。锁机制主要是执行对多个活动事务的并发控制。它可以控制多个用户对同一数据进行的操作，使用锁机制，可以解决数据库的并发问题。本章将介绍事务与锁相关的内容，主要有事务的原理与管理常用语句、事务的类型和应用、锁的作用与类型、锁的应用等。

- 了解什么是事务
- 掌握管理事务的常用语句
- 掌握设置保存点的方法
- 掌握事务的应用案例
- 了解什么是锁
- 掌握锁的分类
- 掌握等待锁和死锁的发生过程
- 熟练掌握综合案例中死锁的应用案例

13.1 事务管理

事务是 Oracle 中的基本工作单元，它是用户定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的工作单位。本节将学习事务的管理方法。

13.1.1 什么是事务

事务是一组包含一条或多条语句的逻辑单元，在事务中的语句被作为一个整体，要么被一起提交，作用在数据库上，使数据库数据被修改；要么被一起撤销，对数据库不做任何的修改。

例如常见的银行账号之间转账的例子，主要操作分 3 步完成：第一步在源账号中减少转账金额，例如减少 1 万；第二步在目标账号中增加转账金额，增加 1 万；第三步在事务日志中记录该事务。

在上面的 3 步操作中，如果有一步失败，整个事务都会回滚，所有的操作都将撤销，目标账号和源账号上的金额都不会发生变化。

13.1.2 事务的属性

事务是作为单个逻辑工作单元执行的一系列操作。一个逻辑工作单元必须有四个属性，称为原子性（Atomic）、一致性（Consistent）、隔离性（Isolated）和持久性（Durable）属性，简称 ACID 属性，只有这样才能成为一个事务。

（1）原子性：事务必须是原子工作单元；对于其数据修改，要么全都执行，要么全都不执行。

（2）一致性：事务在完成时，必须使所有的数据都保持一致状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构都必须是正确的。

（3）隔离性：由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。事务识别数据时数据所处的状态，要么是另一并发事务修改它之前的状态，要么是第二个事务修改它之后的状态，事务不会识别中间状态的数据。这称为可串行性，因为它能够重新装载起始数据，并且重播一系列事务，以使数据结束时的状态与原始事务执行的状态相同。

（4）持久性：事务完成之后，它对于系统的影响是永久性的。该修改即使出现系统故障也将一直保持。

13.1.3 事务管理的常用语句

Oracle 中常用的事务管理语句包含如下几条：

- SET TRANSACTION——设置事务的属性。
- COMMIT——提交事务。
- SAVEPOINT——设置事务点。
- ROLLBACK——事务失败时执行回滚操作。
- ROLLBACK TO SAVEPOINT——回滚到保存点。

提示

一个事务中可以包含一条语句或者多条语句甚至一段程序，一段程序中也可以包含多个事务。事务可以根据需求把一段事务分成多个组，每个组可以理解为一个事务。

13.1.4 事务的类型

事务的类型分为两种，包括显式事务和隐式事务。

1. 显式事务

显式事务是通过命令完成的，具体语法规则如下：

```
新事务开始
sql statement
...
```



```
COMMIT|ROLLBACK;
```

其中，COMMIT 表示提交事务，ROLLBACK 表示事务回滚。

Oracle 事务不需要设置开始标记。通常有下列情况之一时，事务会开启：

- (1) 登录数据库后，第一次执行 DML 语句。
- (2) 当事务结束后，第一次执行 DML 语句。

2. 隐式事务

隐式事务没有非常明确的开始和结束点，Oracle 中的每一条数据操作语句，例如 SELECT、INSERT、UPDATE 和 DELETE 都是隐式事务的一部分，即使只有一条语句，系统也会把这条语句当作一个事务，要么执行所有语句，要么什么都不执行。

默认情况下，隐式事务 AUTOCOMMIT（自动提交）为打开状态，可以控制提交的状态：

```
SET AUTOCOMMIT ON/OFF
```

当有以下情况出现时，事务会结束：

- (1) 执行 DDL 语句，事务自动提交。比如，使用 CREATE、GRANT 和 DROP 等命令。
- (2) 使用 COMMIT 提交事务，使用 ROLLBACK 回滚事务。
- (3) 正常退出 SQL Plus 是自动提交事务，非正常退出则 ROLLBACK 事务回滚。

13.1.5 事务的应用实例

事务的主要作用是保证数据的一致性。在事务没有提交前，当前会话所做的操作其他会话不会看到。下面通过案例来理解事务的特性。

【例 13.1】理解事务的特性。

为了演示效果，可以创建一个数据表 pt 并插入数据，命令如下：

```
CREATE TABLE pt
(
  id NUMBER(6)
);
INSERT INTO pt VALUES (100);
```

登录 SQL Plus，定义窗口为 SQL Plus1。执行更新操作：

```
UPDATE pt SET id=200;
```

执行成功后，查询表 pt 的内容是否变化，结果如下：

```
SELECT * FROM pt;

ID
```

```
-----
200
```

以同样的用户登录新的 SQL Plus，定义窗口为 SQL Plus2，同样查询表 pt 的内容，结果如下：

```
SELECT * FROM pt;

   ID
-----
 100
```

从结果可知，当会话 1 还没有提交时，会话 2 不能看到修改的数据。

在 SQL Plus1 窗口中提交事务，命令如下：

```
COMMIT;
```

执行完成后提示如下：

```
提交完成
```

再次在窗口 SQL Plus2 中查询表 pt 的内容，结果如下：

```
SELECT * FROM pt;

   ID
-----
 200
```

13.1.6 事务的保存点

在事务中可以根据实际的工作需要设置保存点，保存点可以设置在任何位置，当然也可以设置多个保存点，这样就可以把一个长的事务根据需要划分为多个小的段，这样操作的好处是当对数据的操作出现问题时不需要全部回滚，只需要回滚到保存点即可。

事务可以回滚保存点以后的操作，但是保存点会被保留，保存点以前的操作不会回滚。下面仍然通过一个案例来理解保存点的应用。

向数据表 pt 中插入数据，此时隐式事务已经自动打开，命令如下：

```
INSERT INTO pt VALUES (300);
```

创建保存点，名称为 BST，命令如下：

```
SAVEPOINT BST;
```

保存点创建成功后，提示信息如下：

```
保存点已创建
```


继续向数据表 pt 中插入数据，命令如下：

```
INSERT INTO pt VALUES (400);
```

此时查看 pt 表中的记录，结果如下：

```
SELECT * FROM pt;

  ID
-----
 200
 300
 400
```

回滚到保存点 BST，命令如下：

```
ROLLBACK TO BST;
```

此时查看 pt 表中的记录，结果如下：

```
SELECT * FROM pt;

  ID
-----
 200
 300
```

从结果可以看出，保存点以后的操作被回滚，保存点以前的操作被保留。

事务开始之后，事务中所有的操作都会写到事务日志中。写到日志中的事务，一般有两种：一种是针对数据的操作，例如插入、修改和删除，这些操作的对象是大量的数据；另一种是针对任务的操作，例如创建索引。当取消这些事务操作时，系统自动执行这种操作的反操作，保证系统的一致性。系统自动生成一个检查点机制，这个检查点周期地检查事务日志，如果在事务日志中，事务全部完成，那么检查点将事务日志中的事务提交到数据库中，并且在事务日志中做一个检查点提交标识；如果在事务日志中，事务没有完成，那么检查点不将事务日志中的事务提交到数据库中，并且在事务日志中做一个检查点未提交的标识。

13.2 锁

Oracle 支持多用户共享同一数据库，但是，当多个用户对同一个数据库进行修改时，会产生并发问题。使用锁可以解决用户存取数据的问题，从而保证数据库的完整性和一致性。



13.2.1 什么是锁

在多个会话同时操作一个表时，需要防止事务之间的破坏性交互，此时就需要对优先操作的会话进行锁定。可见，锁是在共享资源中控制访问的一种机制。

从事务的分离性可以看出，当前事务不能影响其他的事务，所以当多个会话访问相同的资源时，数据库会利用锁确保它们像队列一样依次进行。Oracle 处理数据时用到的锁是自动获取的，但是 Oracle 也允许用户手动锁定数据。对于一般的用户，通过系统的自动锁管理机制基本可以满足使用要求，但如果对数据安全、数据库完整性和一致性有特殊要求，则需要亲自控制数据库的锁和解锁，这就需要了解 Oracle 的锁机制，掌握锁的使用方法。

如果不使用锁机制，对数据的并发操作会带来下面一些问题：脏读、幻读、非重复性读取、丢失更新。

1. 脏读

当一个事务读取的记录是另一个事务的一部分时，如果第一个事务正常完成，就没有什么问题，如果此时另一个事务未完成，就产生了脏读。例如，员工表中编号为 1001 的员工工资为 1740，如果事务 1 将工资修改为 1900，但还没有提交确认；此时事务 2 读取员工的工资为 1900；事务 1 中的操作因为某种原因执行了 ROLLBACK 回滚，取消了对员工工资的修改，但事务 2 已经把编号为 1001 的员工的数据读走了。此时就发生了脏读。如果此时用了行级锁，第一个事务修改记录时封锁改行，那么第二个事务只能等待，这样就避免了脏数据的产生，从而保证了数据的完整性。

2. 幻读

当某一数据行执行 INSERT 或 DELETE 操作，而该数据行恰好属于某个事务正在读取的范围时，就会发生幻读现象。例如，现在要对员工涨工资，将所有低于 1700 的工资都涨到新的 1900，事务 1 使用 UPDATE 语句进行更新操作，事务 2 同时读取这一批数据，但是在其中插入了几条工资小于 1900 的记录，此时事务 1 如果查看数据表中的数据，会发现自己 UPDATE 之后还有工资小于 1900 的记录！幻读事件是在某个凑巧的环境下发生的，简而言之，它是在运行 UPDATE 语句的同时执行了 INSERT 操作。因为插入了一个新记录行，所以没有被锁定，并且能正常运行。

3. 非重复性读取

如果一个事务不止一次地读取相同的记录，但在两次读取中间有另一个事务刚好修改了数据，则两次读取的数据将出现差异，此时就发生了非重复读取。例如，事务 1 和事务 2 都读取一条工资为 2310 的数据行，如果事务 1 将记录中的工资修改为 2500 并提交，而事务 2 使用的员工的工资仍为 2310。

4. 丢失更新

一个事务更新了数据库之后，另一个事务再次对数据库更新，此时系统只能保留最后一个



数据的修改。

例如对一个员工表进行修改，事务 1 将员工表中编号为 1001 的员工工资修改为 1900，而之后事务 2 又把该员工的工资更改为 3000，那么最后员工的工资为 3000，导致事务 1 的修改丢失。

使用锁将可以实现并发控制，能够保证多个用户同时操作同一数据库中的数据而不发生上述数据不一致的现象。

13.2.2 锁的分类

Oracle 中提供了两种锁模式，包括排他锁和共享锁。

(1) 排他锁：用于数据修改操作，例如 INSERT、UPDATE 或 DELETE，以确保不会同时对同一资源进行多重更新。

(2) 共享锁：用于读取数据操作，允许多个事务读取相同的数据，但不允许其他事务修改当前数据，如 SELECT 语句。当多个事务读取一个资源时，资源上存在共享锁，任何其他事务都不能修改数据，除非将事务隔离级别设置为可重复读或者更高的级别，或者在事务生存周期内用锁定提示对共享锁进行保留，那么一旦数据完成读取，资源上的共享锁立即得以释放。

13.2.3 锁的类型

Oracle 按对象的不同，使用不同类型的锁来管理并发会话对数据对象的操作，从而使数据库实现高度的并发访问。

按照操作对象，锁分为以下几种类型：

- DML 锁：该类型的锁主要是为了保护数据，也称为数据锁。
- DDL 锁：用于保护模式中对象的结构。
- 内部锁：完全自动调用，主要保护数据库的内部结构。

DML 锁主要保证了并发访问数据的完整性，可以分为行级锁和表级锁。

(1) 行级锁：属于排他锁，也被称为事务锁。当修改表的记录时，需要对将要修改的记录添加行级锁，防止两个事务同时修改相同的记录，事务结束后，该锁也会释放。

(2) 表级锁：主要作用是防止在修改表的数据时，表的结构发生变化。

下面通过案例进行验证操作。

在 SQL Plus 中修改表 pt 的记录，命令如下：

```
UPDATE pt SET id=800 WHERE id=300;
```

此时已经锁定了该表，在事务结束前，不允许其他会话对表 pt 进行 DDL 操作。

打开另一个 SQL Plus 窗口，对表进行 DDL 操作，命令如下：

```
DROP TABLE pt;
```



结果如下：

错误报告：

SQL 错误：ORA-00054：资源正忙，但指定以 NOWAIT 方式获取资源，或者超时失效

在 Oracle 中除了执行 DML 时自动为表添加锁外，用户还可以手动添加锁。添加锁的语法规则如下：

```
LOCK TABLE [schema.] table IN  
    [EXCLUSIVE]  
    [SHARE]  
    [ROW EXCLUSIVE]  
    [SHARE ROW EXCLUSIVE]  
    [ROW SHARE*| SHARE UPDATE*]  
    MODE[NOWAIT]
```

如果要释放锁，只需要使用 ROLLBACK 命令即可。

13.2.4 锁等待和死锁

由排他锁机制可知，当一个会话正在修改表的某个记录时，会对该记录进行加锁，如果此时另外一个会话也来修改此记录，会因为等不到排他锁而一直等待，数据库长时间没有响应，直到第一个会话把事务提交，释放排他锁后，第二个会话才能对数据进行修改操作。

下面通过一个案例来理解锁等待。

打开 SQL Plus 窗口，修改 pt 表中 id 为 200 的记录，命令如下：

```
UPDATE pt SET id=600 WHERE id=200;
```

如果修改成功，提示如下：

已更新1行

此时虽然更新操作已经完成，但是事务还没有提交。

打开另外一个 SQL Plus 窗口，同样修改 pt 表中 id 为 200 的记录，命令如下：

```
UPDATE pt SET id=800 WHERE id=200;
```

此时的执行效果不会提示已更新，而是一直等待。主要原因是第一个会话封锁了该记录，事务还没有结束，锁不会释放，而第二个会话得不到该记录的排他锁，所以只能等待。

在两个或多个任务中，如果每个任务锁定了其他任务试图锁定的资源，此时会造成这些任务永久阻塞，从而出现死锁。此时系统处于死锁状态。

1. 死锁的原因

在多用户环境下，死锁的发生是由于两个事务都锁定了不同的资源的同时又都在申请对



方锁定的资源，即一组进程中的各个进程均占有不会释放的资源，但因互相申请其他进程占用的不会释放的资源而处于一种永久等待的状态。形成死锁有 4 个必要条件：

- (1) 请求与保持条件——获取资源的进程可以同时申请新的资源。
- (2) 非剥夺条件——已经分配的资源不能从该进程中剥夺。
- (3) 循环等待条件——多个进程构成环路，并且其中每个进程都在等待相邻进程正占用的资源。
- (4) 互斥条件——资源只能被一个进程使用。

2. 可能会造成死锁的资源

每个用户会话可能有一个或多个代表它运行的任务，其中每个任务可能获取或等待获取各种资源。以下类型的资源可能会造成阻塞，并最终导致死锁：

(1) 锁。等待获取资源（如对象、页、行、元数据和应用程序）的锁可能导致死锁。例如，事务 T1 在行 r1 上有共享锁（S 锁）并等待获取行 r2 的排他锁（X 锁），事务 T2 在行 r2 上有共享锁（S 锁）并等待获取行 r1 的排他锁（X 锁）。这将导致一个锁循环，其中，T1 和 T2 都等待对方释放已锁定的资源。

(2) 工作线程。排队等待可用工作线程的任务可能导致死锁。如果排队等待的任务拥有阻塞所有工作线程的资源，则将导致死锁。例如，会话 S1 启动事务并获取行 r1 的共享锁（S 锁）后，进入睡眠状态。在所有可用工作线程上运行的活动会话正尝试获取行 r1 的排他锁（X 锁）。因为会话 S1 无法获取工作线程，所以无法提交事务并释放行 r1 的锁，这将导致死锁。

(3) 内存。当并发请求等待获得内存，而当前的可用内存无法满足其需要时，可能发生死锁。例如，两个并发查询（Q1 和 Q2）作为用户定义函数执行，分别获取 10MB 和 20MB 的内存。如果每个查询需要 30MB 而可用总内存为 20MB，则 Q1 和 Q2 必须等待对方释放内存，这将导致死锁。

(4) 并行查询执行的相关资源。通常与交换端口关联的处理协调器、发生器或使用者线程至少包含一个不属于并行查询的进程时，可能会相互阻塞，从而导致死锁。此外，当并行查询启动执行时，Oracle 将根据当前的工作负荷确定并行度或工作线程数。如果系统工作负荷发生意外更改，例如，当新查询开始在服务器中运行或系统用完工作线程时，则可能发生死锁。

3. 减少死锁的策略

复杂的系统中不可能百分之百避免死锁，从实际出发，为了减少死锁，可以采用以下策略：

- (1) 在所有事务中以相同的次序使用资源。
- (2) 使事务尽可能简短并且在一个批处理中。
- (3) 为死锁超时参数设置一个合理范围，如 3~30 分钟。超时，则自动放弃本次操作，避免进程挂起。
- (4) 避免在事务内和用户进行交互，减少资源的锁定时间。



13.3 综合案例——死锁的案例

死锁是锁等待的一个特例，通常发生在两个或者多个会话之间。下面通过案例来理解死锁的发生过程。

打开第一个 SQL Plus 窗口，修改表 pt 中 id 字段为 200 的记录，命令如下：

```
UPDATE pt SET id=600 WHERE id=200;
```

打开第二个 SQL Plus 窗口，修改表 pt 中 id 字段为 300 的记录，命令如下：

```
UPDATE pt SET id=800 WHERE id=300;
```

目前，第一个会话锁定了 id 字段为 200 的记录，第二个会话锁定了 id 字段为 300 的记录。第一个会话修改第二个会话已经修改的记录，命令如下：

```
UPDATE pt SET id=600 WHERE id=300;
```

此时第一个会话出现了锁等待，因为它修改的记录被第二个会话锁定。

第二个会话修改第一个会话修改的记录，命令如下：

```
UPDATE pt SET id=800 WHERE id=200;
```

此时会出现死锁的情况。Oracle 会自动检测死锁的情况，并释放一个冲突锁，把消息传给对方事务。此时第一个会话窗口中提示检测到死锁，信息如下：

错误报告：

SQL 错误：ORA-00068：等待资源时检测到死锁

此时 Oracle 自动做出处理，重新回到锁等待的情况。

13.4 疑难解惑

疑问 1：事务和锁应用上的区别是什么？

事务将一段语句作为一个单元来处理，这些操作要么全部成功，要么全部失败。事务包含四个特性：原子性、一致性、隔离性和持久性。事务的方式分为显式事务和隐式事务。事务以“COMMIT”或“ROLLBACK”语句结束。锁是另一个和事务紧密联系的概念，对于多用户系统，使用锁来保护指定的资源。在事务中使用锁，防止其他用户修改另外一个还没有完成的事务中的数据。SQL Server 中有多种类型的锁，允许事务锁定不同的资源。

疑问 2：事务和锁有什么关系？

在 Oracle 中可以使用多种机制来确保数据的完整性，例如约束、触发器以及本章介绍的



事务和锁等。事务和锁的关系非常紧密。事务包含一系列的操作，这些操作要么全部成功，要么全部失败，通过事务机制管理多个事务，保证事务的一致性。在事务中使用锁保护指定的资源，防止其他用户修改另外一个还没有完成的事务中的数据。

13.5 经典习题

- (1) 简述事务的原理。
- (2) 事务都有哪些类型？
- (3) 为什么会产生死锁？
- (4) 常用的锁类型有哪些？



第 14 章

◀ Oracle 的用户管理 ▶

Oracle 是一个多用户数据库，具有功能强大的访问控制系统，可以为不同用户指定允许的权限。用户管理包括管理用户账户、权限等。本章将向读者介绍 Oracle 用户管理中的相关知识，包括：账户管理、权限管理、角色管理和概要文件。

- 掌握账户管理的方法
- 掌握权限管理的方法
- 掌握角色管理的方法
- 掌握管理概要文件的方法

14.1 账户管理

Oracle 提供了许多语句，这些语句可以用来管理包括创建用户、删除用户、密码管理和权限管理等内容。Oracle 数据库的安全性，需要通过账户管理来保证。本节将介绍如何对账户进行管理。

14.1.1 管理账号概述

根据每个用户访问 Oracle 数据库的需求不同，Oracle 需要赋予用户不同的操作权限，这是数据库管理员经常遇到的问题。如果管理员对用户权限分配不合理，将会对数据库造成一定的安全隐患。

Oracle 中用户登录数据库的方式主要有 3 种。

- (1) 密码验证方式：把验证密码放在 Oracle 数据库中，这是最常用的验证方式，同时安全性也比较高。
- (2) 外部验证方式：这种验证的密码通常与数据库所在的操作系统的密码一致。
- (3) 全局验证方式：这种验证方式也不把密码放在 Oracle 数据库中，也是不常用的验证方式。



14.1.2 新建普通用户

创建新用户，必须有相应的权限来执行创建操作。在 Oracle 数据库中，创建用户时需要特别注意用户的密码必须以字母开头。

用户可以使用 CREATE USER 语句创建用户。语法规则如下：

```
CREATE USER username IDENTIFIED BY password
OR EXTERNALLY AS certificate_DN
OR GLOBALLY AS directory_DN
[DEFAULT TABLESPACE tablespace]
[TEMPORARY TABLESPACE tablespace| tablespace_group_name]
[QUOTA size|UNLIMITED ON tablespace]
[PROFILE profile]
[PASSWORD EXPIRE]
[ACCOUNT LOCK|UNLOCK]
```

username 表示创建的用户的名称；IDENTIFIED BY password 表示以口令作为验证方式；EXTERNALLY AS certificate_DN 表示以外部验证方式；GLOBALLY AS directory_DN 表示以全局验证方式；DEFAULT TABLESPACE 表示设置默认表空间，如果忽略该语句，那么创建的用户就存在数据库的默认表空间中，如果数据库没有设置默认表空间，那么创建的用户就放在 SYSTEM 表空间中；TEMPORARY TABLESPACE 设置临时表空间或者临时表空间组，可以把临时表空间存放在临时表空间组中，如果忽略该语句，那么就会把临时文件存放到当前数据库默认的临时表空间中，如果没有默认的临时表空间，那么就会把临时文件存放到 SYSTEM 的临时表空间中；QUOTA 表示设置当前用户使用表空间的最大值，在创建用户时可以有多个 QUOTA 来设置用户在不同表空间中能够使用的表空间大小，如果设置成 UNLIMITED，表示对表空间的使用没有限制；PROFILE 设置当前用户使用的概要文件的名称，如果忽略了该子句，那么该用户就使用当前数据库中默认的概要文件；PASSWORD EXPIRE 用于设置当前用户密码立即处于过期状态，用户如果想再次登录数据库必须更改密码；ACCOUNT 用于设置锁定状态，如果设置成 LOCK，那么用户不能访问数据库，如果设置成 UNLOCK，那么用户可以访问数据库。

【例 14.1】以口令验证的方式，使用 CREATE USER 创建一个用户，用户名是 USER01，密码是 mypass，并且设置成密码立即过期的方式，实现代码如下：

```
CREATE USER USER01
IDENTIFIED BY mypass
DEFAULT TABLESPACE mytest
QUOTA 20M ON mytest
TEMPORARY TABLESPACE mytemp
PROFILE my_test
```




```
PASSWORD EXPIRE;
```

上述代码创建了用户 USER01，密码为 mypass，用户 USER01 的默认表空间为 mytest，用户可以在表空间 mytest 中使用的磁盘限额为 20MB，用户 USER01 的临时表空间是 mytemp，用户 USER01 的概要文件是 my_test，并且用户的密码是立即过期状态。

【例 14.2】以外部验证的方式，使用 CREATE USER 创建一个用户，用户名是 USER02，实现代码如下：

```
CREATE USER USER02  
IDENTIFIED EXTERNALLY  
DEFAULT TABLESPACE mytest  
QUOTA 10M ON mytest  
PROFILE my_test;
```

上述代码创建了用户 USER02，验证方式为外部验证，用户 USER02 的默认表空间为 mytest，用户可以在表空间 mytest 中使用的磁盘限额为 10MB，用户 USER02 的概要文件是 my_test。

14.1.3 修改用户信息

在 Oracle 数据库中，可以使用 ALTER USER 语句修改用户信息。具体使用的语法规则如下：

```
ALTER USER username IDENTIFIED  
{ BY password[REPLACE old_password]  
| EXTERNALLY [AS certificate_DN]  
| GLOBALLY [AS directory_DN]  
}  
[DEFAULT TABLESPACE tablespace]  
[TEMPORARY TABLESPACE tablespace| tablespace_group_name]  
[QUOTA size|UNLIMITED ON tablespace]  
[PROFILE profile]  
[PASSWORD EXPIRE]  
[ACCOUNT LOCK|UNLOCK]
```

上面的各个参数的含义和创建用户的参数含义一样，这里就不再重复讲述。

【例 14.3】修改 USER01 的密码为 newpassword，实现代码如下：

```
ALTER USER USER01  
IDENTIFIED BY newpassword  
DEFAULT TABLESPACE mytest;
```




【例 14.4】修改 USER01 的临时表空间为 newmytemp，实现代码如下：

```
ALTER USER USER01  
TEMPORARY TABLESPACE new mytemp;
```

【例 14.5】设置 USER01 的密码为立即过期，实现代码如下：

```
ALTER USER USER01  
PASSWORD EXPIRE;
```

14.1.4 删除用户

在 Oracle 数据库中，可以使用 DROP USER 语句删除用户，具体的语法规则如下：

```
DROP USER username [CASCADE];
```

username 为用户的名称；关键字 CASCADE 是可选参数，如果要删除的用户中没有任何数据库对象，可以省略 CASCADE 关键字。

【例 14.6】使用 DROP USER 删除用户 USER01，实现代码如下：

```
DROP USER USER01 CASCADE;
```

提示

DROP USER 不能自动关闭任何打开的用户对话。而且，如果用户有打开的对话，此时取消用户，命令则不会生效，直到用户对话被关闭后才能生效。一旦对话被关闭，用户也被取消，此用户再次试图登录时将会失败。

14.2 权限管理

权限管理主要是对登录到 Oracle 的用户进行权限验证。所有用户的权限都存储在 Oracle 的权限表中，不合理的权限规划会给 Oracle 服务器带来安全隐患。数据库管理员要对所有用户的权限进行合理规划管理。Oracle 权限系统的主要功能是证实连接到一台给定主机的用户，并且赋予该用户在数据库上的 SELECT、INSERT、UPDATE 和 DELETE 权限。本节将为读者介绍 Oracle 权限管理的内容。

14.2.1 授权

授权就是为某个用户授予权限。合理的授权可以保证数据库的安全。Oracle 中可以使用 GRANT 语句为用户授予权限。

在 Oracle 中，必须是拥有 GRANT 权限的用户才可以执行 GRANT 语句。授予权限包括授予系统权限和授予对象权限。



授予系统权限的语法如下：

```
GRANT system_privilege
|ALL PRIVILEGES TO {user IDENTIFIED BY password|role|}
[WITH ADMIN OPTION]
```

其中，`system_privilege` 表示创建的系统权限名称；`ALL PRIVILEGES` 表示可以设置除 `SELECT ANY DICTIONARY` 权限以外的所有系统权限；`{user IDENTIFIED BY password|role|}` 表示设置权限的对象，`role` 代表的是设置角色的权限；`WITH ADMIN OPTION` 表示当前给予授权的用户还可以给其他用户进行系统授权的赋予。

【例 14.7】使用 `GRANT` 语句为用户 `USER01` 赋予一个系统权限 `session`，实现代码如下：

```
GRANT create session to USER01
```

【例 14.8】使用 `GRANT` 语句为用户 `USER02` 赋予一个系统权限 `session`，并且该用户也有授予 `create session` 的权限，实现代码如下：

```
GRANT create session to USER02 WITH ADMIN OPTION
```

授予对象权限的语法规则如下：

```
GRANT object_privilege|ALL
ON schema.object
TO user|role
[WITH ADMIN OPTION]
[WITH THE GRANT ANY OBJECT]
```

其中，`object_privilege` 表示创建的对象权限名称；如果选择 `ALL`，则代表授予用户所有的对象权限，这个权限在使用的时候一定要注意；`schema.object` 表示为用户授予的对象权限使用的对象；`user|role` 中的 `user` 代表是用户，`role` 代表是角色；`WITH ADMIN OPTION` 表示当前给予授权的用户还可以给其他用户进行系统授权的赋予；`WITH THE GRANT ANY OBJECT` 表示当前给予授权的用户还可以给其他用户进行对象授权的赋予。

【例 14.9】使用 `GRANT` 语句为用户 `USER01` 赋予表对象 `FRT` 更新的权限，实现代码如下：

```
GRANT UPDATE ON FRT TO USER01
```

14.2.2 收回权限

收回权限就是取消已经赋予用户的某些权限。收回用户不必要的权限可以在一定程度上保证系统的安全性。Oracle 中使用 `REVOKE` 语句取消用户的某些权限。

收回权限包括收回系统权限和收回对象权限。

只有数据库管理员才能收回系统权限，而且撤销系统权限的前提是当前的用户已经存在要撤销的系统权限。收回系统权限的语法规则如下：


```
REVOKE system_privilege
FROM user|role
```

【例 14.10】使用 REVOKE 语句收回用户 USER01 的系统权限 session，实现代码如下：

```
REVOKE create session FROM USER01
```

使用 REVOKE 语句也可以收回对象权限。具体的语法规则如下：

```
REVOKE object_privilege|ALL
ON schema.object
FROM user|role
[CASCADE CONSTRAINTS]
```

[CASCADE CONSTRAINTS]选项表示该用户授予其他用户的权限也一并收回。

【例 14.11】使用 REVOKE 语句收回用户 USER02 在 FRT 对象上的更新权限，实现代码如下：

```
REVOKE UPDATE ON FRT FROM USER02
```

提示

收回系统权限和收回对象权限有不同的地方。如果撤销用户的系统权限，那么该用户授予其他用户的系统权限仍然存在；如果撤销用户的对象权限，那么该用户授予其他用户的对象权限也被收回。

14.2.3 查看权限

在 Oracle 中，用户的权限存放在数据库的数据字典中，用户的系统权限存放在数据字典 DBA_SYS_PRIVS 中，用户的对象权限存放在数据字典 DBA_TAB_PRIVS 中。数据库管理员可以通过用户名查看用户的权限。

【例 14.12】查看 ANONYMOUS 用户的系统权限，实现代码如下：

```
SELECT * FROM DBA_SYS_PRIVS WHERE GRANTEE='ANONYMOUS';
```

查询结果如下：

GRANTEE	PRIVILEGE	ADMIN_OPTION	COMMON
ANONYMOUS	CREATE SESSION	NO	YES

【例 14.13】查看 ANONYMOUS 用户的对象权限，实现代码如下：

```
SELECT PRIVILEGE FROM DBA_TAB_PRIVS WHERE GRANTEE='ANONYMOUS';
```

查询结果如下：

```
PRIVILEGE
-----
UPDATE
SELECT
INSERT
DELETE
EXECUTE
```

如果想查看系统中所有用户的名称等信息，可以使用下列命令之一：

```
SELECT * FROM DBA_USERS;
SELECT * FROM ALL_USERS;
SELECT * FROM USER_USERS;
```

14.3 角色管理

角色相当于 Windows 操作系统中的用户组，可以集中管理数据库或服务器的权限。

14.3.1 角色概述

数据库角色是针对某个具体数据库的权限分配，数据库用户可以作为数据库角色的成员，继承数据库角色的权限，数据库管理人员也可以通过管理角色的权限来管理数据库用户的权限。

用户和角色是不同的，用户是数据库的使用者，角色是权限的授予对象，给用户授予角色，相当于给用户授予一组权限。数据库中的角色可以授予多个用户，一个用户也可以被授予多个角色。角色是数据库中管理员定义的权限集合，可以方便地对不同用户进行权限授予。

例如，创建一个具有插入权限的角色，那么被赋予这个角色的用户，就具备了插入的权限。

14.3.2 创建角色

在实际的数据库管理过程中，通过创建角色，可以分组管理用户的权限。下面将介绍角色的创建过程。

创建角色的具体语法如下：

```
CREATE ROLE role
[NOT IDENTIFIED| IDENTIFIED BY[ password]| IDENTIFIED BY
```



```
EXTERNALLY| IDENTIFIED BY GLOBALLY]
```

NOT IDENTIFIED 表示创建角色的验证方式为不需要验证；IDENTIFIED BY[password] 表示创建角色的验证方式为口令验证；IDENTIFIED BY EXTERNALLY 表示创建角色的验证方式为外部验证；IDENTIFIED BY GLOBALLY 表示创建角色的验证方式为全局验证。

【例 14.14】创建角色 MYROLE，实现代码如下：

```
CREATE ROLE MYROLE
NOT IDENTIFIED;
```

角色创建完成后，即可对角色赋予权限，具体语法格式如下：

```
GRANT system_privilege
|ALL PRIVILEGES TO role
[WITH ADMIN OPTION]
```

【例 14.15】赋予 MYROLE 角色 CREATE SESSION 权限，实现代码如下：

```
GRANT CREATE SESSION TO MYROLE;
```

给角色授予权限时，数据库管理员必须拥有 GRANT_ANY_PRIVILEGES 权限才可以给角色赋予任何权限。

14.3.3 设置角色

角色创建完成后不能直接使用，还需要把角色赋予用户才能使角色生效。将角色赋予用户的具体语法如下：

```
GRANT role TO user
```

【例 14.16】将角色 MYROLE 赋予 USER01，实现代码如下：

```
GRANT MYROLE TO USER01;
```

一个用户可以同时被赋予多个角色，被赋予的多个角色是否生效可以自行设置。设置的方法如下：

```
SET ROLE role
SET ROLE ALL
SET ROLE ALL EXCEPT role
SET ROLE NONE
```

其中，SET ROLE role 表示指定的角色生效；SET ROLE ALL 表示设置用户的所有角色都生效；SET ROLE ALL EXCEPT role 表示设置 EXCEPT 后的角色不失效；SET ROLE NONE 表示设置用户的角色都失效。

【例 14.17】设置角色 MYROLE 在当前用户上生效，实现代码如下：

```
SET ROLE MYROLE;
```

也可以通过以下代码实现：

```
SET ROLE ALL EXCEPT MYROLE;
```

14.3.4 修改角色

角色创建完成后，还可以修改其内容。具体的语法规则如下：

```
ALTER ROLE role
[NOT IDENTIFIED| IDENTIFIED BY[ password]| IDENTIFIED BY
EXTERNALLY| IDENTIFIED BY GLOBALLY]
```

上面的代码只能修改角色本身，如果想修改已经赋予角色的权限或者角色，则要使用 GRANT 或者 REVOKE 来完成。

14.3.5 查看角色

用户可以查询数据库中已经存在的角色，也可以查询指定用户的角色的相关信息。

【例 14.18】查询 SYSTEM 用户的角色，实现代码如下：

```
SELECT GRANTED_ROLE, DEFAULT_ROLE FROM DBA_ROLE_PRIVS
WHERE GRANTEE='SYSTEM';
```

结果如下：

GRANTED_ROLE	DEFAULT_ROLE
-----	-----
AQ_ADMINISTRATOR_ROLE	YES
DBA	YES
MGMT_USER	YES

14.3.6 删除角色

对于不再需要的角色，可以删除。在删除角色的同时，所有拥有该角色的用户也将自动撤销该角色所授予的权限。

删除角色的语法格式如下：

```
DROP ROLE role
```


【例 14.19】使用 DROP ROLE 删除角色 MYROLE，语句如下：

```
DROP ROLE MYROLE;
```

14.4 管理概要文件 PROFILE

Oracle 数据库中的概要文件为 PROFILE，它为数据库的管理带来极大的便利，本章节将讲述概要文件的相关操作。

14.4.1 PROFILE 概述

PROFILE 就是 Oracle 数据库中的概要文件，主要用于存放数据库中的系统资源或者数据库限制使用的内容。默认情况下，如果用户没有创建概要文件，则使用系统的默认概要文件，名称为 DEFAULT。

概要文件会给数据库管理员带来很大的便利，数据库管理员可以先对数据库中的用户分组，根据每一组的权限不同，建立不同的概要文件，这样便于管理用户。

值得注意的是，概要文件只能用于用户，不能在角色中使用。

14.4.2 创建概要文件

数据库中默认的概要文件为 PROFILE，根据实际的需要，可以创建概要文件。

创建概要文件的语法格式如下：

```
CREATE PROFILE profile
LIMIT
{resource_parameters|password_parameters}
```

resource_parameters 表示资源参数，主要包括如下：

- CPU_PER_SESSION: 表示一个会话占用 CPU 的总量。
- CPU_PER_CALL: 表示允许一个调用占用 CPU 的最大值。
- CONNECT_TIME: 代表运行一个持续的会话的最大值。

password_parameters 表示口令参数，主要包括如下：

- PASSWORD_LIFE_TIME: 指的是多少天后口令失效。
- PASSWORD_REUSE_TIME: 指密码保留的时间。
- PASSWORD_GRACE_TIME: 指设置密码失效后锁定。

【例 14.20】创建一个概要文件 MYPROFILE，设置密码保留天数为 80 天。实现代码如下：

```
CREATE PROFILE MYPROFILE
```

```
LIMIT
PASSWORD_REUSE_TIME 80;
```

14.4.3 修改概要文件

使用 ALTER PROFILE 语句可以修改已经存在的概要文件，语法格式如下：

```
ALTER PROFILE profile
LIMIT
{resource_parameters|password_parameters}
```

【例 14.21】修改概要文件 MYPROFILE，设置 CONNECT_TIME 为 2000。实现代码如下：

```
ALTER PROFILE MYPROFILE
LIMIT
CONNECT_TIME 2000;
```

14.4.4 删除概要文件

对于不需要的概要文件，可以对其执行删除操作。具体的语法格式如下：

```
DROP PROFILE profile [CASCADE]
```

如果删除的概要文件已经被用户使用过，那么删除概要文件时要加上 CASCADE 关键词，这样用户所使用的概要文件也被撤销；如果概要文件没有被使用过，可以省略该关键词。

【例 14.22】使用 DROP PROFILE 删除概要文件，语句如下：

```
DROP PROFILE MYPROFILE CASCADE;
```

在 Oracle 中，默认的概要文件 PROFILE 是不能删除的。

14.5 疑难解惑

疑问 1：如何查询已经存在的概要文件？

概要文件被保存在数据字典 DBA_PROFILES 中，如果想查询概要文件，可以使用如下语句：

```
SELECT * FROM DBA_PROFILES;
```

疑问 2：角色如何继承？

一个角色可以继承其他角色的权限集合。例如，角色 MYROLE 具备对表 fruits 的增加、

删除权限，此时创建一个新的角色 MYROLE01，该角色继承角色 MYROLE 的权限，实现的语句如下：

```
GRANT MYROLE TO MYROLE01;
```

14.6 经典习题

创建数据库 Team，定义数据表 player，语句如下：

```
CREATE TABLE player
{
    playid    NUMBER(6) PRIMARY KEY,
    playname  VARCHAR2(30) NOT NULL,
    teamnum   NUMBER(6) NOT NULL UNIQUE,
    info      VARCHAR2(50)
};
```

执行以下操作：

- (1) 创建一个新账户，用户名为 account1，密码为 oldpwd1。授权该用户对数据库中 player 表的 SELECT 和 INSERT 权限，并且授权该用户对 player 表的 info 字段的 UPDATE 权限。
- (2) 更改 account1 用户的密码为 newpwd2。
- (3) 查看授权给 account1 用户的权限。
- (4) 收回 account1 用户的权限。
- (5) 将 account1 用户的账号信息从系统中删除。

第 15 章

◀ 控制文件和日志 ▶

Oracle 控制文件主要用来存放数据库的名字、数据库的位置等信息。日志记录了 Oracle 数据库日常操作。控制文件和日志文件都存储了 Oracle 数据库中的重要信息。本章主要讲述控制文件和日志文件的使用方法和技巧。

- 了解什么是控制文件
- 了解什么是 Oracle 日志
- 掌握应用控制文件的方法
- 掌握管理日志文件的方法

15.1 控制文件简介

控制文件是数据库中最小的文件，是一个二进制文件，其中包括了数据库的结构信息，同时也包括了数据文件和日志文件的一些信息。控制文件虽小，但可以说是 Oracle 中最重要的文件，只有 Oracle 进程才能够更新控制文件中的内容。控制文件中主要包括数据库名称、位置、联机或者脱机状态、Redo Log File 的位置和名称、表空间名称、Archive Log File 信息、CheckPoint 信息、Undo 信息、RMAN 信息等，从控制文件中包含的内容也可以看出控制文件在整个 Oracle 中的重要性。

控制文件在每个数据库中都存在，但是一个控制文件只能属于一个数据库。这就像是工作证，每个员工都有工作证，但是一个工作证只能属于一个员工。在创建数据库时，控制文件被自动创建，如果数据库的信息发生变化，控制文件也会随之改变。控制文件不能手动修改，只能由 Oracle 数据库本身来修改。控制文件在数据库启动和关闭时都要使用，如果没有控制文件，数据库将无法工作。

【例 15.1】在数据字典中查看控制文件的信息。实现代码如下：

```
desc v$controlfile;
```


运行结果如下：

名称	空值	类型
-----	-----	-----
STATUS		VARCHAR2 (7)
NAME		VARCHAR2 (513)
IS_RECOVERY_DEST_FILE		VARCHAR2 (3)
BLOCK_SIZE		NUMBER
FILE_SIZE_BKLS		NUMBER
CON_ID		NUMBER

在数据字典前面加 v\$, 表示是当前实例的动态视图。从结果可以看出, 控制文件的数据字典就是一组表和视图结构, 存放数据库所用的有关信息, 对用户来说是一组只读的表。

15.2 控制文件的应用案例

本节主要讲述控制文件的一些应用操作和技巧。

15.2.1 查看控制文件的内容

通过数据字典 v\$controlfile, 可以查看控制文件的存放位置和状态。

【例 15.2】在数据字典中查看控制文件的存放位置和状态。实现代码如下：

```
SELECT name,status FROM v$controlfile;
```

运行结果如下：

NAME	STATUS
-----	-----
F:\APP\TEST\ORADATA\ORCL\CONTROL01.CTL	
F:\APP\TEST\ORADATA\ORCL\CONTROL02.CTL	

15.2.2 更新控制文件的内容

当数据文件出现增加、重命名和删除等操作时, Oracle 服务器会立刻更新控制文件以反映数据库结构的这种变化。每次在数据库的结构发生变化后, 为了防止数据丢失都要备份控制文件。各进程根据分工的不同分别把更改后的数据库信息写入到控制文件中: 日志写入进程负责把当前日志序列号记录到控制文件中; 校验点进程负责把校验点的信息记录到控制文件中; 归档进程负责把归档日志的信息记录到控制文件中。

为了应对磁盘损坏等数据灾难的情况，用户可以把控制文件进行镜像操作，这样即使一个文件被破坏，其他的控制文件依然存在，数据也不会丢失，数据库还可以正常运行。

15.2.3 使用 init.ora 多路复用控制文件

控制文件虽然由数据库直接创建，但是在数据库初始化之前，用户可以修改初始化文件 init.ora。要修改 init.ora，需要先找到它的存放位置，这个文件的位置在安装目录的 admin\orcl\pfile 下，如图 15-1 所示。

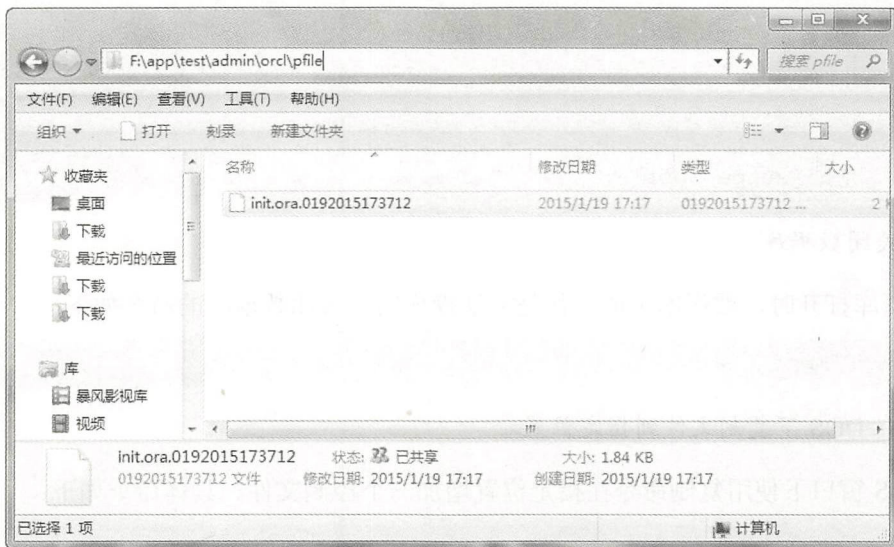


图 15-1 init.ora 的位置

在修改 init.ora 文件之前，先通过复制操作把控制文件复制到不同的位置，然后用记事本打开 init.ora 文件，找到 control_files 参数后即可进行修改，如果 15-2 所示。修改时需要注意，在每个控制文件之间是通过逗号分隔的，并且每一个控制文件都是用双括号括起来的。在修改控制文件的路径之前，需要把控制文件复制一份进行保存，以免数据库无法启动。

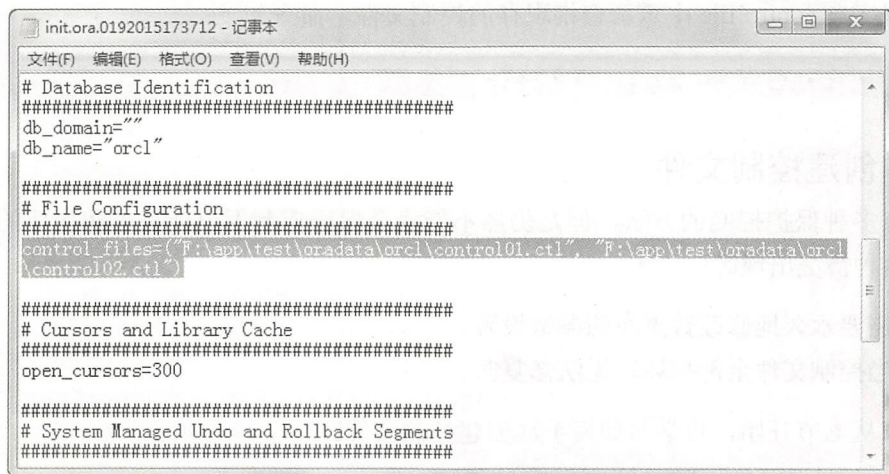


图 15-2 修改 init.ora 文件

15.2.4 使用 SPFILE 多路复用控制文件

除了修改 init.ora 初始化参数的方式可以实现多路复用控制文件外，还可以通过 SPFILE 方式实现多路复用，它们的原理和修改参数是一样的。

【例 15.3】使用 SPFILE 多路复用控制文件。具体操作步骤如下：

步骤 01 修改 control_files 参数。

在确保数据库是打开状态时，使用以下命令修改 control_files 参数，语句如下：

```
ALTER SYSTEM SET control_files='文件的路径1',  
'文件的路径2',  
'文件的路径3', ...,  
'文件的路径 n' scope=spfile;
```

步骤 02 关闭数据库。

在数据库打开时，数据库中的文件是无法操作的。关闭数据库的命令如下：

```
shutdown immediate;
```

步骤 03 在 DOS 下复制文件到指定位置。

在 DOS 窗口下使用复制命令在指定位置增加一个控制文件。具体命令如下：

```
copy 旧文件, 新文件
```

步骤 04 启动数据库实例并验证。

文件复制完成后，使用 startup 命令重新启动数据库。

```
Startup;
```

在数据字典 controlfile 中重新查询现存的控制文件，命令如下：

```
SELECT name, status FROM v$controlfile;
```

15.2.5 创建控制文件

虽然有多种保护控制的方法，但是仍然不能完全保证控制不出现丢失和损坏的情况。特别是以下两种情况出现时：

- (1) 需要永久地修改数据库的参数设置。
- (2) 当控制文件全部损坏，无法修复时。

为此，从本节开始，将学习如何手动创建控制文件。

【例 15.4】手动创建控制文件。具体操作步骤如下：

步骤 01 找原有的数据文件和重做日志的路径。

获取数据文件和重做日志路径的方法有两种：一种方法是，当控制文件没有损坏时，从控制文件中直接获取；另外一种方法是，如果控制文件损坏了，从数据字典 `v$datafile` 中获取数据文件的信息，从数据字典 `v$logfile` 中获取日志文件的信息。当然，使用上面两种方法的前提是数据库能够正常启动。如果数据库不能正常启动，那么需要根据系统的错误信息来查找原因。在创建新的控制文件并且使用它打开数据库以后，Oracle 会对数据字典和控制文件的内容进行检查。如果发现数据字典包含了某个数据文件而控制文件没有列出这个数据文件，Oracle 数据库就会报错。数据库管理员可以根据这些信息来判断是否缺少必要的数据库文件，一步步查找到真正的数据文件。

获取数据文件的命令如下：

```
SELECT name FROM v$datafile;
```

查询结果如下：

```
NAME
-----
F:\APP\TEST\ORADATA\ORCL\SYSTEM01.DBF
F:\APP\TEST\ORADATA\ORCL\PDBSEED\SYSTEM01.DBF
F:\APP\TEST\ORADATA\ORCL\SYSAUX01.DBF
F:\APP\TEST\ORADATA\ORCL\PDBSEED\SYSAUX01.DBF
F:\APP\TEST\ORADATA\ORCL\UNDOTBS01.DBF
F:\APP\TEST\ORADATA\ORCL\USERS01.DBF
F:\APP\TEST\ORADATA\ORCL\PDBORCL\SYSTEM01.DBF
F:\APP\TEST\ORADATA\ORCL\PDBORCL\SYSAUX01.DBF
F:\APP\TEST\ORADATA\ORCL\PDBORCL\SAMPLE_SCHEMA_USERS01.DBF
F:\APP\TEST\ORADATA\ORCL\PDBORCL\EXAMPLE01.DBF
F:\APP\TEST\PRODUCT\12.1.0\DBHOME_1\DATABASE\MYTEM.DBF
```

获取日志文件的命令如下：

```
SELECT member FROM v$logfile;
```

结果如下：

```
MEMBER
-----
F:\APP\TEST\ORADATA\ORCL\REDO03.LOG
F:\APP\TEST\ORADATA\ORCL\REDO02.LOG
```



```
F:\APP\TEST\ORADATA\ORCL\REDO01.LOG
```

步骤 02 关闭数据库。

在创建控制文件之前，需要先关闭数据库。命令如下：

```
shutdown immediate;
```

为了保证数据库的安全，关闭数据库后，应该把数据库的日志文件、数据库文件、参数文件等备份到其他硬盘上。

步骤 03 创建新的控制文件。

把原来的控制文件备份到其他位置后，还需要启动一个数据库实例。启动实例的语句如下：

```
startup nomount;
```

参数 `nomount` 表示只启动实例。

启动实例后就可以创建控制文件了。创建控制文件的语句如下：

```
create controlfile
reuse database '数据库实例名' noresetlogs //是否重做日志或重命名数据库 noarchivelog //
归档状态
maxlogfiles //最大日志文件大小
maxlogmembers //日志文件组的成员数
maxinstances //最大实例的个数
maxloghistory //最大历史日志文件个数
logfile //日志文件
group1 '日志文件的路径1' size 日志文件的大小,
...
groupn '日志文件的路径n' size 日志文件的大小
datafile //数据文件
'路径1',
...
'路径n'
character set we8dec
```

参数 `noresetlogs` 表示在创建控制文件时不需要重做日志文件和重命名数据库，否则可以使用 `resetlogs` 参数。

步骤 04 使用 SPFILE 方法修改 init.ora 中 control_files 参数。

**步骤 05 验证控制文件。**

重启数据库后，查询 v\$controlfile 数据字典，检查控制文件是否全部正确加载。如果数据库加载不了，可以重新启动数据库服务。验证的命令如下：

```
SELECT name FROM v$controlfile;
```

至此，控制文件创建成功。

15.3 日志简介

Oracle 日志主要分为两类 包括重做日志文件和归档日志文件。重做日志文件是 Oracle 数据库正常运行不可缺少的文件。重做日志文件主要记录数据库操作的过程。在需要恢复数据库时，重做日志文件可以从备份还原的数据库上再执行一次，从而达到数据库的最新状态。

Oracle 系统在运行时有归档模式和非归档模式。在归档模式下，如果重做日志文件全部写满后，就把第一个重做日志文件写入归档日志文件中，再把日志文件写到第一个重做日志文件中，使用归档方式可以方便以后的恢复操作；在非归档模式下，所有的日志文件都写在重做日志文件中，如果重做日志文件写满了，那么就把前面的日志文件覆盖掉。

【例 15.5】在数据字典中查看日志文件的信息。实现代码如下：

```
desc v$logfile;
```

运行结果如下：

名称	空值	类型

GROUP#		NUMBER
STATUS		VARCHAR2 (7)
TYPE		VARCHAR2 (7)
MEMBER		VARCHAR2 (513)
IS_RECOVERY_DEST_FILE		VARCHAR2 (3)
CON_ID		NUMBER

在归档模式下，Oracle 的性能会受到一定的影响，所以 Oracle 默认情况下采用的是非归档模式。获取当前 Oracle 的归档模式可以从 v\$database 数据字典中查看。

【例 15.6】查看 v\$database 数据字典中的描述内容。实现代码如下：

```
desc v$database;
```




如果需要查看当前数据库的模式，可以通过查看当前数据库的 `log_mode` 的值实现。

【例 15.7】查看当前数据库的模式。实现代码如下：

```
SELECT name, log_mode FROM v$database;
```

查询结果如下：

NAME	LOG_MODE
ORCL	NOARCHIVELOG

从结果可以看出，当前模式为非归档模式。如果结果为 `ARCHIVELOG`，则表示当前模式为归档模式。

15.4 管理日志文件

在 Oracle 数据库中，日志文件全部存放在日志文件组中。本节将讲述日志文件的管理方法。

15.4.1 新建日志文件组

通过日志文件组，数据库管理员可以轻松地管理日志文件。创建日志文件组的语法如下：

```
ALTER DATABASE [database_name]
ADD LOGFILE GROUP n
filename SIZE m;
```

其中，参数 `database_name` 为要修改的数据库名，如果省略，表示为当前数据库；参数 `n` 为创建日志工作组的组号，组号在日志组中必须是唯一的；参数 `filename` 表示日志文件组的存在位置；参数 `m` 表示日志文件组的大小，默认情况下大小为 50MB。

【例 15.8】新建日志文件组。实现代码如下：

```
ALTER DATABASE
ADD LOGFILE GROUP 6
('F:\app\test\oradata\orcl\mylogn.log') SIZE 20M;
```

执行结果如下：

```
database add LOGFILE 已变更。
```

可见，数据库中已经创建了新的日志文件组。



15.4.2 添加日志文件到日志文件组

添加日志文件和添加日志文件组的语法非常类似。语法规则如下：

```
ALTER DATABASE [database_name]
ADD LOGFILE MEMBER
filename TO GROUP n;
```

其中，参数 `database_name` 为要修改的数据库名，如果省略，则表示为当前数据库；参数 `filename` 表示日志文件的存在位置；参数 `n` 为日志文件填入的组号。

【例 15.9】添加日志文件到日志文件组。实现代码如下：

```
ALTER DATABASE
ADD LOGFILE MEMBER
'F:\app\test\oradata\orcl\mylog.log'
TO GROUP 6;
```

执行结果如下：

```
database add LOGFILE 已变更。
```

此时创建的日志文件添加到日志文件组 6 中，添加的日志文件名称为 `mylog.log`。

15.4.3 删除日志文件组和日志文件

使用 `ALTER DATABASE` 语句可以删除日志文件组，具体的语法规则如下：

```
ALTER DATABASE [database_name]
DROP LOGFILE
GROUP n;
```

其中，参数 `n` 为日志文件组的组号。

【例 15.10】删除日志文件组 6。实现代码如下：

```
ALTER DATABASE
DROP LOGFILE
GROUP 6;
```

执行结果如下：

```
database drop LOGFILE 已变更。
```

此时，日志文件组 6 被成功删除。

删除日志文件的方法与删除日志文件组的方法类似。语法格式如下：



```
ALTER DATABASE [database_name]
DROP LOGFILE MEMBER
filename;
```

其中，参数 `filename` 表示日志文件的名称，当然也包括日志文件的路径。

【例 15.11】删除日志文件 `mylog.log`。实现代码如下：

```
ALTER DATABASE
DROP LOGFILE MEMBER
'F:\app\test\oradata\orcl\mylog.log';
```

执行结果如下：

```
database drop LOGFILE 已变更。
```

此时，日志文件 `mylog.log` 被成功删除。

15.4.4 查询日志文件组和日志文件

用户可以查询已经存在的日志文件组和日志文件。

查询日志文件组主要是通过 `V$LOG` 来实现，下面通过案例来讲解具体的方法。

【例 15.12】查询 `V$LOG` 中的组号（`GROUP#`）、成员数（`MEMBERS`）和状态（`STATUS`）的信息。实现代码如下：

```
SELECT GROUP#, MEMBERS, STATUS FROM V$LOG;
```

执行结果如下：

GROUP#	MEMBERS	STATUS
1	1	CURRENT
2	1	INACTIVE
3	1	INACTIVE
6	1	INACTIVE

查询日志文件主要通过 `V$LOGFILE` 来实现，下面通过案例来讲解具体的方法。

【例 15.13】查询 `V$LOGFILE` 中的组号（`GROUP#`）、成员（`MEMBER`）的信息。实现代码如下：

```
SELECT GROUP#, MEMBERS FROM V$LOGFILE;
```

执行结果如下：



GROUP#	MEMBER
3	F:\APP\TEST\ORADATA\ORCL\REDO03.LOG
2	F:\APP\TEST\ORADATA\ORCL\REDO02.LOG
1	F:\APP\TEST\ORADATA\ORCL\REDO01.LOG
6	F:\APP\TEST\ORADATA\ORCL\MYLOGN.LOG

15.5 疑难解惑

疑问 1：联机日志文件的状态有哪些？

在 Oracle 日志文件中，最容易模糊的就是日志文件的 3 个状态，它们的含义如下：

- current: 表示 LGWR 正在写的日志文件。
- active: 表示 LGWR 正在写的日志文件，实例恢复时需要这种文件。
- inactive: 表示 LGWR 正在写的日志文件，实例恢复时也不会用到这种文件。

疑问 2：如何提高日志的切换频率？

通过参数 ARCHIVE_LAG_TARGET 可以控制日志切换的时间间隔，以秒为单位。通过减少时间间隔，从而实现提高日志的切换频率。例如以下代码：

```
SQL> ALTER SYSTEM SET ARCHIVE_LAG_TARGET=50 SCOPE=both;
```

通过上面的命令，可以实现日志每 50 秒切换一次。

15.6 经典习题

- (1) 创建控制文件，然后验证是否成功创建。
- (2) 控制文件的多路复用怎么操作？
- (3) 创建日志文件组。
- (4) 创建日志文件、删除日志文件。



第 16 章

◀ 数据备份与还原 ▶

尽管采取了一些管理措施来保证数据库的安全，但是不确定的意外情况总是有可能造成数据的损失，例如意外的停电、管理员不小心的操作失误都可能会造成数据的丢失。保证数据库安全的最重要的一个措施是确保对数据进行定期备份。如果数据库中的数据丢失或者出现错误，可以使用备份的数据进行还原，这样就尽可能地降低了意外原因导致的损失。Oracle 提供了多种方法对数据进行备份和还原。本章将介绍数据备份、数据还原和数据导入导出的相关知识。

- 了解什么是数据备份
- 掌握各种数据备份的方法
- 掌握各种数据还原的方法
- 掌握表的导入和导出方法

16.1 数据备份

数据备份是数据库管理员非常重要的工作。系统意外崩溃或者硬件的损坏都可能导致数据的丢失，因此 Oracle 管理员应该定期地备份数据，使得在意外情况发生时，尽可能减少损失。

本节将介绍数据备份的两种方法。

16.1.1 冷备份

冷备份也就是脱机备份，当数据库正常关闭后，通过脱机备份可将关键性文件复制到另外的存储位置，脱机备份是一种快速、安全的备份方法。

冷备份的优点：

- (1) 备份快速、安全、简便。
- (2) 可快速执行时间点恢复。
- (3) 能与归档方法相结合，做数据库“最佳状态”的恢复。
- (4) 对备份文件维护简单、安全。



冷备份的不足：

- (1) 单独使用时，只能将数据库恢复到备份时的状态。
- (2) 备份过程中数据库必须处于脱机状态，对数据库要求较高的业务势必造成损失。
- (3) 只能进行物理备份，对存储介质造成空间浪费。
- (4) 恢复过程中只能进行完整数据库恢复，不能进行更小粒度的恢复。

冷备份需要备份的文件如下：

- (1) 所有数据文件。
- (2) 所有控制文件。
- (3) init.ora 文件等参数文件。

【例 16.1】冷备份数据库。

首先正常关闭数据库，使用以下 3 行命令之一即可：

```
shutdown immediate;  
shutdown transactional;  
shutdown normal;
```

然后通过操作系统命令或者手动复制文件到指定位置，此时需要较大的介质存储空间。

16.1.2 热备份

热备份也称为联机备份，在数据库的归档模式下进行备份。

【例 16.2】查看数据库中日志的状态。

```
archive log list;
```

查询结果如下：

数据库日志模式	不归档模式
自动归档	已禁用
归档目标	USE_DB_RECOVERY_FILE_DEST
最早的联机日志序列	208
当前日志序列	211

从结果可以看出，目前数据库的日志模式是不归档模式，同时自动模式也是禁用的。

【例 16.3】设置数据库日志模式为归档模式。实现代码如下：

```
alter system set log_archive_start=true scope=spfile;  
shutdown immediate;
```




Oracle 12c 从零开始学（视频教学版）

```
startup mount;  
alter database archive log;
```

其中，第一行的作用是修改系统的日志方式为归档模式；第二行作用是关闭数据库；第三行作用是启动 mount 实例，但是不启动数据库；第四行作用是更改数据库为归档模式。

设置完成后，再次查询当前数据库的日志模式，命令如下：

```
archive log list;
```

查询结果如下：

数据库日志模式	归档模式
自动归档	启用
归档目标	USE_DB_RECOVERY_FILE_DEST
最早的联机日志序列	208
当前日志序列	211

从结果可以看出，当前日志模式已经修改为归档模式，并且自动存档已经启动。

把数据库设置成归档模式后，就可以进行数据库的备份与恢复操作了。

【例 16.4】备份表空间 MYTEM。

改变数据库的状态为 open，命令如下：

```
alter database open;
```

备份表空间 MYTEM，开始命令如下：

```
alter tablespace MYTEM begin backup;
```

打开数据库中的 oradata 文件夹，把文件复制到磁盘中的另外一个文件夹或其他磁盘上。

结束备份命令如下：

```
alter tablespace MYTEM end backup;
```

16.2 数据还原

管理人员操作的失误、计算机故障以及其他意外情况，都会导致数据的丢失和破坏。当数据丢失或意外破坏时，可以通过还原已经备份的数据尽量减少数据丢失和破坏造成的损失。本节将介绍数据还原的方法。

【例 16.5】恢复表空间 MYTEM 中的数据文件。

对当前的日志进行归档，命令如下：



```
alter system archive log current;
```

一般情况下，一个数据库中包含 3 个日志文件，所以需要使用 3 次下面的语句来切换日志文件：

```
alter system switch logfile;
```

关闭数据库服务，命令如下：

```
shutdown immediate;
```

删除数据文件并重新启动数据库。首先找到数据库文件的存放位置，默认情况下，数据库文件存放在数据库的 `oradata` 文件夹中。如果不清楚数据库文件的存放位置，可以在 `V$datafile` 数据字典中查看表空间中数据文件的位置，找到数据文件后删除即可，然后启动数据库。启动数据库的命令如下：

```
startup;
```

此时如果缺少数据库文件，会显示错误信息。

在恢复数据文件之前，先把数据文件设置成脱机状态，并且删除该数据文件。命令如下：

```
alter database datafile 6 offline drop;
```

把数据库设置成 `open` 状态，命令如下：

```
alter database open;
```

在数据库的状态是 `open` 时就可以恢复数据库了。恢复表空间 `MYTEM` 的数据文件命令如下：

```
recover datafile 6;
```

这里的 6 是数据文件的编号。

数据恢复完成后，设置数据文件为联机状态，命令如下：

```
alter database datafile 6 online;
```

至此，数据文件恢复完成。在恢复数据库中的数据时，把数据库文件设置成脱机状态后，就需要把之前备份好的数据复制到原来的数据文件存放的位置，否则会提示错误。

16.3 表的导出和导入

有时会将 Oracle 数据库中的数据导出到外部存储文件中，Oracle 数据库中的数据表可以导出，同样这些导出文件也可以导入到 Oracle 数据库中。本小节将介绍数据导出和导入的常用方法。



16.3.1 用 EXP 工具导出数据

在 DOS 窗口下，输入以下语句，然后根据提示导出表即可。

```
C:\> EXP username/password
```

其中，`username` 为登录数据库的用户名；`password` 为用户密码。注意：这里的用户不能为 `SYS`。

【例 16.6】导出数据表 `books`，代码如下：

```
C:\> EXP system/ Manager123 file=f: \mytest.dmp tables=books;
```

这里指出了导出文件的名称和路径，然后指出导出表的名称。如果要导出多个表，可以在各个表之间用逗号隔开。

导出表空间和导出表不同，导出表空间的用户必须是数据库的管理员角色。导出表空间的命令如下：

```
C:\> EXP username/password FILE=filename.dmp TABLESPACES=tablespaces_name
```

其中，参数 `username/password` 表示具有数据库管理员权限的用户名和密码；`filename.dmp` 表示存放备份的表空间的数据文件；`tablespaces_name` 表示要备份的表空间名称。

【例 16.7】导出表空间 `MYTEM`，代码如下：

```
C:\> EXP system/ Manager123 file=f: \mytest01.dmp TABLESPACES=MYTEM
```

16.3.2 用 EXPDP 导出数据

EXPDP 是从 Oracle 10g 开始提供的导入导出工具，采用的是数据泵技术，该技术是在数据库之间或者数据库与操作系统之间传输数据的工具。

数据泵技术的主要特性如下：

- (1) 支持并行处理导入、导出任务。
- (2) 支持暂停和重启动导入、导出任务。
- (3) 支持通过联机的方式导出或导入远端数据库中的对象。
- (4) 支持在导入时实现导入过程中自动修改对象属主、数据文件或数据所在表空间。
- (5) 导入/导出时提供了非常细粒度的对象控制，甚至可以详细制定是否包含或不包含某个对象。

下面开始讲述使用 EXPDP 导出数据的过程。

1. 创建目录对象

使用 EXPDP 工具之前，必须创建目录对象，具体的语法规则如下：

```
SQL> CREATE DIRECTORY directory_name AS 'file_name';
```



其中，参数 `directory_name` 为创建目录的名称；`file_name` 表示存放数据的文件夹名。

【例 16.8】创建目录对象 MYDIR，代码如下：

```
SQL> CREATE DIRECTORY MYDIR AS 'DIRMP';
```

结果如下：

```
directory MYDIR 已创建。
```

2. 给使用目录的用户赋权限

新创建的目录对象不是所有用户都可以使用，只有拥有该目录权限的用户才可以使用。假设备份数据库的用户是 SCOTT，那么赋予权限的具体语法如下：

```
SQL> GRANT READ,WRITE ON DIRECTORY directory_name TO SCOTT;
```

其中，参数 `directory_name` 表示目录的名称。

【例 16.9】将目录对象 MYDIR 权限赋予 SCOTT，代码如下：

```
SQL> GRANT READ,WRITE ON DIRECTORY MYDIR TO SCOTT;
```

运行结果如下：

```
赋权成功。
```

3. 导出指定的表

创建完目录后，即可使用 EXPDP 工具导出数据，操作也是在 DOS 的命令窗口中完成。指定备份表的语法格式如下：

```
C:\> EXP username/password DIRECTORY= directory_name DUMPFILE= file_name  
TABLE=table_name;
```

其中，参数 `directory_name` 表示存放导出数据的目录名称；`file_name` 表示导出数据存放的文件名；`table_name` 表示准备导出的表名，如果导出多个表，可以用逗号隔开。

【例 16.10】导出数据表 BOOKS，代码如下：

```
C:\> EXP scott/tiger DIRECTORY= MYDIR DUMPFILE=mytemp.dmp TABLE=BOOKS;
```

16.3.3 用 IMP 导入数据

逻辑上导入数据和导出数据是逆过程，使用 EXP 导出的数据，可以使用 IMP 导入。

【例 16.11】使用 EXP 导入 fruits 表，命令如下：

```
C:\> EXP scott/tiger file=f: \mytest2.dmp tables=fruits;
```




【例 16.12】使用 IMP 导入 fruits 表，命令如下：

```
C:\> IMP scott/tiger file= mytest2.dmp tables=fruits;
```

16.3.4 用 IMPDP 导入数据

使用 EXPDP 导出数据后，可以使用 IMPDP 将数据导入。

【例 16.13】使用 IMPDP 导入 BOOKS 表，命令如下：

```
C:\>IMPDP scott/tiger DIRECTORY= MYDIR DUMPFILE=mytemp.dmp TABLE=BOOKS;
```

如果数据库中 BOOKS 表已经存在，此时会报错，解决方式是在上面代码后加上：ignore=y

16.4 疑难解惑

疑问 1：如何判断数据导出是否成功？

在做导出操作时，无论是否成功，都会有提示信息。常见的信息的含义如下：

(1) 导出成功，没有任何错误，将会提示如下的信息。

```
Export terminated successfully without warnings
```

(2) 导出完成，但是某些对象有问题，将会提示如下的信息。

```
Export terminated successfully with warnings
```

(3) 导出失败，将会提示如下的信息。

```
Export terminated unsuccessfully
```

疑问 2：如何把数据导出到磁带上？

Oracle 的导出工具 EXP 支持把数据直接备份到磁带上，这样可以减少把数据备份到本地磁盘，然后再备份到磁带上的中间环节。命令如下：

```
EXP system/ Manager123 file=/dev/rmt0 tables=books;
```

其中，参数 file 指定的就是磁带的设备名。



16.5 经典习题

- (1) 使用冷备份的方法备份数据文件。
- (2) 使用热备份的方法备份数据文件。
- (3) 还原备份的数据。
- (4) 使用各种方法导出和导入数据文件。



第 17 章

◀ 性能优化 ▶

Oracle 性能优化就是通过合理安排资源、调整系统参数使 Oracle 运行更快、更节省资源。Oracle 性能优化包括查询速度优化、更新速度优化、Oracle 服务器优化等。本章将为读者讲解以下内容：性能优化的介绍、查询优化、数据库结构优化、Oracle 服务器优化。

- 了解什么是优化
- 掌握优化查询的方法
- 掌握优化数据库结构的方法
- 掌握优化 Oracle 服务器的方法

17.1 优化简介

优化 Oracle 数据库是数据库管理员和数据库开发人员的必备技能。Oracle 优化，一方面是找出系统的瓶颈，提高 Oracle 数据库整体的性能；另一方面需要合理的结构设计和参数调整，以提高用户操作响应的速度；同时还要尽可能节省系统资源，以便系统可以提供更大负荷的服务。本节将为读者介绍优化的基本知识。

Oracle 数据库优化是多方面的，原则是减少系统的瓶颈，减少资源的占用，提高系统的反应速度。例如，通过优化文件系统，提高磁盘 I/O 的读写速度；通过优化操作系统调度策略，提高 Oracle 在高负荷情况下的负载能力；优化表结构、索引、查询语句等使查询响应更快。

众所周知，从内存中直接读取数据的速度远远大于从磁盘中读取数据。然而影响内存读取速度的因素有两个，包括内存的大小和内存的分配、使用和管理方法。由于 Oracle 提供了自动内存管理机制，所以用户只需要手动分配内存即可。Oracle 中的内存主要包括两部分：系统全局区和进程全局区。它们既可以在数据库启动时进行加载，也可以在数据库使用中进行设置。下面将详细讲述这两部分的具体管理方法。

17.1.1 修改系统全局区

系统全局区，简称为 SGA，是 System Global Area 的缩写。SGA 是共享的内存机构，主要存储的是数据库的公用信息，因此 SGA 也被称为共享全局区。SGA 主要包括共享池、缓冲



区、大型池、Java 池和日志缓冲区。

【例 17.1】查看当前数据库的 SGA 状态，命令如下：

```
SQL> show parameter sga;
```

查询结果如下：

NAME	TYPE	VALUE
lock_sga	boolean	FALSE
pre_page_sga	boolean	TRUE
sga_max_size	big integer	1648M
sga_target	big integer	0
unified_audit_sga_queue_size	integer	1048576

其中，需要注意的结果有两个：`sga_max_size` 和 `sga_target`。`sga_max_size` 是为 SGA 分配的最大内存，`sga_target` 指定的是数据库可管理的最大内存。如果 `sga_target` 值为 0，表示关闭共享内存区。

在 Oracle 中，管理员还可以通过视图 `v$sgastat` 来查看 SGA 的具体分配情况。命令如下：

```
SQL> SELECT * FROM v$sgastat;
```

客户端接收到查询语句后，首先进行语法分析，然后是语义分析，最后才是执行步骤。在执行步骤以前的操作就是 SQL 语句的预处理，这些预处理都是在共享池中进行缓存，缓存的标示是根据 SQL 语句所形成的 Hash 值。服务器收到 SQL 语句是根据 Hash 值在共享池中查找是否已经有预处理的 SQL 语句，如果存在，则直接进行数据库操作，否则将进行语法分析。对于共享池来说，存在命中率的概念，也就是直接从共享池中获取执行计划的成功率。成功率越高，代表数据库的性能越高。因此，共享池命中率是影响 SQL 语句的重要指标。

NAMESPACE	GETHITRATIO	PINHITRATIO
SQL AREA	0.4170824773	0.8984711785
TABLE/PROCEDURE	0.7304022819	0.7962924099
BODY	0.8535031847	0.8787401575
TRIGGER	0.6666666667	0.6666666667
INDEX	0.1353383459	0.1407407407
CLUSTER	0.9551569507	0.9611650485
DIRECTORY	0.8125	0.8125
QUEUE	0.6666666667	0.6



RULESET	0	0.6666666667
SUBSCRIPTION	0	0
EDITION	0.9950248756	0.9946380697
DBLINK	0.9864864865	1
OBJECT ID	0	1
SCHEMA	0.995688726	1
DBINSTANCE	0	1
SQL AREA STATS	0.05980528512	0.05980528512
SQL AREA BUILD	0.1088339223	1
PDB	0.5	1
AUDIT POLICY	0.98	0.98
PDBOPER	0.1666666667	1

其中，GETHITRATIO 为 SQL 语句解析时，直接获得解释计划的命中率；PINHITRATIO 是执行命中率。

【例 17.2】修改 SGA 内存大小，命令如下：

```
SQL> alter system set sga_max_size=2000M scope=spfile;
```

结果如下：

```
system SET 已变更。
```

其中，scope=spfile 表示设置作用到数据库启动文件中，一旦数据库重启，则该参数将立即重启。

修改参数 sga_target 为 2000MB，代码如下：

```
SQL> alter system set sga_target =2000M scope=spfile;
```

结果如下：

```
system SET 已变更。
```

数据库重启后，SGA 的大小已经被成功修改了。

17.1.2 修改进程全局区

进程全局区简称为 PGA。每个客户端连接到 Oracle 服务器都由服务器分配一定内存来保持连接，并将在该内存中实现用户私有操作。所有用户连接的内存集合就是 Oracle 数据库的 PGA。



【例 17.3】查看 PGA 的状态，命令如下：

```
show parameter pga;
```

查询结果如下：

NAME	TYPE	VALUE
-----	-----	-----
pga_aggregate_limit	big integer	2G
pga_aggregate_target	big integer	0

参数，`pga_aggregate_target` 可以指定 PGA 内存的最大值。当 `pga_aggregate_target` 值大于 0 时，Oracle 将自动管理 PGA 内存。

【例 17.4】修改 PGA 的大小，命令如下：

```
SQL>alter system set pga_aggregate_target=500M scope=both;
```

结果如下：

```
system SET 已变更。
```

其中，`scope=both` 表示同时修改当前环境与启动文件 `spfile`。

17.2 优化查询

查询是数据库中最频繁的操作，提高查询速度可以有效地提高 Oracle 数据库的性能。本节将为读者介绍优化查询的方法。

17.2.1 分析查询语句的执行计划

如果想要分析 SQL 语句的性能，可以查看该语句的执行计划，从而分析每一步执行是否存在问题。

查看执行计划的方法有以下两种。

1. 通过设置 AUTOTRACE 查看执行计划

设置 AUTOTRACE 的具体含义如下：

- (1) SET AUTOTRACE OFF：此为默认值，即关闭 AUTOTRACE。
- (2) SET AUTOTRACE ON EXPLAIN：只显示执行计划。
- (3) SET AUTOTRACE ON STATISTICS：只显示执行的统计信息。
- (4) SET AUTOTRACE ON：包含 (2)、(3) 两项的内容。



(5) SET AUTOTRACE TRACEONLY: 与(4)相似, 但不显示语句的执行结果。

【例 17.5】通过设置 AUTOTRACE 查看执行计划, 命令如下:

```
SQL> set autotrace on;
```

运行上述命令后结果如下:

已启用自动跟踪

显示执行计划以及语句的统计信息。

执行查询语句, 命令如下:

```
SQL> select * from fruits;
```

结果如下:

f_id	s_id	f_name	f_price
a1	101	apple	5.20
a2	103	apricot	2.20
b1	101	blackberry	10.20
b2	104	berry	7.60
b5	107	xxxx	3.60
bs1	102	orange	11.20
bs2	105	melon	8.20
c0	101	cherry	3.20
l2	104	lemon	6.40
m1	106	mango	15.60
m2	105	xbabay	2.60
m3	105	xxtt	11.60
o2	103	coconut	9.20
t1	102	banana	10.30
t2	102	grape	5.30
t4	107	xbababa	3.60

选定了 16 行

Plan hash value: 1063410116


```

-----
| Id | Operation          | Name | Rows | Bytes | Cost (%CPU)| Time |
-----
|  0 | SELECT STATEMENT   |      |  16 |  2592 |      2  (0)| 00:00:01 |
|  1 | TABLE ACCESS FULL | FRUITS |  16 |  2592 |      2  (0)| 00:00:01 |
-----

```

Note

```
-----
- dynamic statistics used: dynamic sampling (level=2)
```

Statistics

```
-----
0 user rollbacks
0 global enqueue gets async
0 physical read total IO requests
0 physical read partial requests
23 consistent gets pin
0 gc current blocks served
0 gc current block pin time
0 gc current blocks pinned
0 gc cr blocks received
0 gc cr block receive time

```

在查询结果中，ID 表示一个序号，但不是执行的先后顺序，执行的先后根据缩进来判断；Operation 表示当前操作的内容；Rows 表示当前操作的 Cardinality，Oracle 估计当前操作的返回结果集；Cost (%CPU) 表示 Oracle 计算出来的一个数值，用于说明 SQL 执行的代价；Time 表示 Oracle 估计当前操作的时间。

2. 使用 EXPLAIN PLAN FOR 语句查看执行计划

使用 EXPLAIN PLAN FOR 语句可以查看执行计划。具体语法格式如下：

```
EXPLAIN PLAN FOR SQL 语句;
```

【例 17.6】通过设置 EXPLAIN PLAN FOR 语句查看执行计划，命令如下：

```
SQL> EXPLAIN PLAN FOR SELECT * FROM fruits;
```


结果如下：

```
plan FOR 成功。
Statistics
-----
0  user rollbacks
0  global enqueue gets async
0  physical read total IO requests
0  physical read partial requests
5  consistent gets pin
0  gc current blocks served
0  gc current block pin time
0  gc current blocks pinned
0  gc cr blocks received
0  gc cr block receive time
```

17.2.2 索引对查询速度的影响

Oracle 中提高性能的一个最有效的方式就是对数据表设计合理的索引。索引提供了高效访问数据的方法，并且加快查询的速度，因此，索引对查询的速度有着至关重要的影响。使用索引可以快速定位表中的某条记录，从而提高数据库查询的速度，提高数据库的性能。本小节将为读者介绍索引对查询速度的影响。

如果查询时没有使用索引，查询语句将扫描表中的所有记录。在数据量大的情况下，这样查询的速度会很慢。如果使用索引进行查询，查询语句可以根据索引快速定位到待查询记录，从而减少查询的记录数，达到提高查询速度的目的。

17.2.3 使用索引查询

索引可以提高查询的速度。但并不是使用带有索引的字段查询时，索引都会起作用。本小节将向读者介绍索引的使用。

使用索引有几种特殊情况，在这些情况下，有可能使用带有索引的字段查询时，索引并没有起作用，下面重点介绍这几种特殊情况。

1. 使用 LIKE 关键字的查询语句

在使用 LIKE 关键字进行查询的查询语句中，如果匹配字符串的第一个字符为“%”，索引不会起作用。只有“%”不在第一个位置，索引才会起作用。

2. 使用多列索引的查询语句

Oracle 可以为多个字段创建索引。一个索引可以包括 16 个字段。对于多列索引，只有查询条件中使用了这些字段中第 1 个字段时，索引才会被使用。

3. 使用 OR 关键字的查询语句

查询语句的查询条件中只有 OR 关键字，且 OR 前后的两个条件中的列都是索引时，查询中才使用索引。否则，查询将不使用索引。

17.2.4 优化子查询

Oracle 支持子查询，使用子查询可以进行 SELECT 语句的嵌套查询，即一个 SELECT 查询的结果作为另一个 SELECT 语句的条件。子查询可以一次性完成很多逻辑上需要多个步骤才能完成的 SQL 操作。子查询虽然可以使查询语句很灵活，但执行效率不高。执行子查询时，Oracle 需要为内层查询语句的查询结果建立一个临时表，然后外层查询语句从临时表中查询记录。查询完毕后，再撤销这些临时表。因此，子查询的速度会受到一定的影响。如果查询的数据量比较大，这种影响就会随之增大。

在 Oracle 中，可以使用连接（JOIN）查询来替代子查询。连接查询不需要建立临时表，其速度比子查询要快，如果查询中使用索引的话，性能会更好。连接之所以更有效率，是因为 Oracle 不需要在内存中创建临时表来完成查询工作。

17.3 优化数据库结构

一个好的数据库设计方案对于数据库的性能常常会起到事半功倍的效果。合理的数据库结构不仅可以使数据库占用更小的磁盘空间，而且能够使查询速度更快。数据库结构的设计，需要考虑数据冗余、查询和更新的速度、字段的数据类型是否合理等多方面的内容。本节将为读者介绍优化数据库结构的方法。

17.3.1 将字段很多的表分解成多个表

对于字段较多的表，如果有些字段的使用频率很低，可以将这些字段分离出来形成新表。因为当一个表的数据量很大时，会由于使用频率低的字段的存在而使查询速度变慢。本小节将为读者介绍这种优化表的方法。

【例 17.7】假设会员表存储会员登录认证信息，该表中有很多字段，如 id、姓名、密码、地址、电话、个人描述字段。其中，地址、电话、个人描述等字段并不常用。可以将这些不常用字段分解出另外一个表，将这个表命名为 members_detail，表中有 member_id、address、telephone、description 等字段。其中，member_id 是会员编号，address 字段存储地址信息，telephone 字段存储电话信息，description 字段存储会员个人描述信息。这样就把会员表分成两

个表，分别为 members 表和 members_detail 表。

创建这两个表的 SQL 语句如下：

```
CREATE TABLE members (
    Id number(11) NOT NULL,
    username varchar2(255) DEFAULT NULL ,
    password varchar2(255) DEFAULT NULL ,
    last_login_time date DEFAULT NULL ,
    last_login_ip varchar2(255) DEFAULT NULL ,
    PRIMARY KEY (id)
) ;

CREATE TABLE members_detail (
    member_id number (11) DEFAULT 0,
    address varchar2(255) DEFAULT NULL ,
    telephone varchar2(16) DEFAULT NULL ,
    description varchar2(255)
) ;
```

这两个表的结构如下：

```
SQL> desc members;

名称          空值      类型
-----
ID             NOT NULL  NUMBER(11)
USERNAME
PASSWORD
LAST_LOGIN_TIME      DATE
LAST_LOGIN_IP        VARCHAR2(255)
```

```
SQL> DESC members_detail;
```

```
名称          空值 类型
-----
MEMBER_ID      NUMBER(11)
ADDRESS        VARCHAR2(255)
TELEPHONE      VARCHAR2(16)
```

```
DESCRIPTION    VARCHAR2(255)
```

如果需要查询会员的详细信息，可以用会员的 id 来查询。如果需要将会员的基本信息和详细信息同时显示，可以将 members 表和 members_detail 表进行联合查询，查询语句如下：

```
SELECT * FROM members LEFT JOIN members_detail ON
members.id=members_detail.member_id;
```

通过这种分解，可以提高表的查询效率。对于字段很多且有些字段使用不频繁的表，可以通过这种分解的方式来优化数据库的性能。

17.3.2 增加中间表

对于需要经常联合查询的表，可以建立中间表以提高查询效率。通过建立中间表，把需要经常联合查询的数据插入到中间表中，然后将原来的联合查询改为对中间表的查询，以此来提高查询效率。本小节将为读者介绍增加中间表优化查询的方法。

首先，分析经常联合查询表中的字段，然后使用这些字段建立一个中间表，并将原来联合查询的表的数据插入到中间表中。最后，就可以使用中间表来进行查询了。

【例 17.8】创建会员信息表和会员组信息表的 SQL 语句如下：

```
CREATE TABLE vip(
    id number(11) NOT NULL,
    username varchar2(255) DEFAULT NULL,
    password varchar2(255) DEFAULT NULL,
    groupId number (11) DEFAULT 0,
    PRIMARY KEY (Id)
);

CREATE TABLE vip_group (
    Id number(11) NOT NULL,
    name varchar2(255) DEFAULT NULL,
    remark varchar2(255) DEFAULT NULL,
    PRIMARY KEY (Id)
);
```

查询会员信息表和会员组信息表的 SQL 语句如下：

```
SQL> DESC vip;
```

```
名称      空值      类型
```

```
-----
```



```
ID          NOT NULL   NUMBER(11)
USERNAME          VARCHAR2(255)
PASSWORD          VARCHAR2(255)
GROUPID          NUMBER(11)
```

```
SQL> DESC vip_group;
```

```
名称      空值      类型
```

```
-----
ID          NOT NULL   NUMBER(11)
NAME          VARCHAR2(255)
REMARK          VARCHAR2(255)
```

已知现在有一个模块需要经常查询带有会员组名称、会员组备注（remark）、会员用户名信息的会员信息。根据这种情况可以创建一个 temp_vip 表。temp_vip 表中存储用户名（user_name）、会员组名称（group_name）和会员组备注（group_remark）信息。创建表的语句如下：

```
CREATE TABLE temp_vip (
    id number (11) NOT NULL,
    user_name varchar2(255) DEFAULT NULL,
    group_name varchar2(255) DEFAULT NULL,
    group_remark varchar2(255) DEFAULT NULL,
    PRIMARY KEY (Id)
);
```

接下来，从会员信息表和会员组表中查询相关信息存储到临时表中：

```
SQL> INSERT INTO temp_vip(user_name, group_name, group_remark)
      SELECT v.username,g.name,g.remark
      FROM vip v ,vip_group g
      WHERE v.groupId =g.Id;
```

以后，可以直接从 temp_vip 表中查询会员名、会员组名称和会员组备注，而不用每次都进行联合查询。这样可以提高数据库的查询速度。

17.3.3 增加冗余字段

设计数据库表时应尽量遵循范式理论的规约，尽可能减少冗余字段，让数据库设计看起来精致、优雅。但是，合理地加入冗余字段可以提高查询速度。本小节将为读者介绍通过增加冗

余字段来优化查询速度的方法。

表的规范化程度越高，表与表之间的关系就越多，需要连接查询的情况也就越多。例如，员工的信息存储在 `staff` 表中，部门信息存储在 `department` 表中。通过 `staff` 表中的 `department_id` 字段与 `department` 表建立关联关系。如果要查询一个员工所在部门的名称，必须从 `staff` 表中查找员工所在部门的编号（`department_id`），然后根据这个编号去 `department` 表查找部门的名称。如果经常需要进行这个操作，连接查询会浪费很多时间。可以在 `staff` 表中增加一个冗余字段 `department_name`，该字段用来存储员工所在部门的名称，这样就不用每次都进行连接操作了。

提示

冗余字段会导致一些问题。比如，冗余字段的值在一个表中被修改了，就要想办法在其他表中更新该字段，否则就会使原本一致的数据变得不一致。分解表、增加中间表和增加冗余字段都浪费了一定的磁盘空间。从数据库性能来看，为了提高查询速度而增加少量的冗余大部分时候是可以接受的。是否通过增加冗余来提高数据库性能，这要根据实际需求综合分析。

17.3.4 优化插入记录的速度

插入记录时，影响插入速度的主要是索引、唯一性校验、一次插入记录条数等。根据这些情况，可以分别进行优化。本小节将为读者介绍优化插入记录速度的几种方法。

对于 MyISAM 引擎的表，常见的优化方法如下：

1. 禁用索引

对于非空表，插入记录时，Oracle 会根据表的索引对插入的记录建立索引。如果插入大量数据，建立索引会降低插入记录的速度。为了解决这种情况，可以在插入记录之前禁用索引，数据插入完毕后再开启索引。禁用索引的语句如下：

```
ALTER index index_name unusable;
```

其中，`index_name` 是禁用索引的名称。

重新开启索引的语句如下：

```
ALTER index index_name usable;
```

2. 禁用唯一性检查

插入数据时，Oracle 会对插入的记录进行唯一性校验。这种唯一性校验也会降低插入记录的速度。为了降低这种情况对查询速度的影响，可以在插入记录之前禁用唯一性检查，等到记录插入完毕后再开启。禁用唯一性检查的语句如下：

```
ALTER TABLE table_name
DISABLE CONSTRAINT constraint_name;
```


其中, table_name 是表的名称, constraint_name 是唯一性约束的名称。

开启唯一性检查的语句如下:

```
ALTER TABLE table_name  
ENABLE CONSTRAINT constraint_name;
```

3. 使用批量插入

插入多条记录时, 可以使用一条 INSERT 语句插入一条记录, 也可以使用一条 INSERT 语句插入多条记录。插入一条记录的 INSERT 语句情形如下:

```
INSERT INTO fruits VALUES('x1', '101 ', 'mongo2 ', '5.6');  
INSERT INTO fruits VALUES('x2', '101 ', 'mongo3 ', '5.6')  
INSERT INTO fruits VALUES('x3', '101 ', 'mongo4 ', '5.6')
```

使用一条 INSERT 语句插入多条记录的情形如下:

```
INSERT INTO fruits VALUES  
SELECT 'x1', '101 ', 'mongo2 ', '5.6' from dual  
Union all  
SELECT 'x2', '101 ', 'mongo3 ', '5.6' from dual  
Union all  
SELECT 'x3', '101 ', 'mongo4 ', '5.6' from dual;
```

第 2 种情形的插入速度要比第 1 种情形快。

17.4 优化 Oracle 服务器

优化 Oracle 服务器主要从两个方面来优化, 一方面是对硬件进行优化; 另一方面是对 Oracle 服务的参数进行优化。这部分的内容需要较全面的知识, 一般只有专业的数据库管理员才能进行这一类的优化。对于可以定制参数的操作系统, 也可以针对 Oracle 进行操作系统优化。本节将为读者介绍优化 Oracle 服务器的方法。

17.4.1 优化服务器硬件

服务器的硬件性能直接决定着 Oracle 数据库的性能。硬件的性能瓶颈, 直接决定 Oracle 数据库的运行速度和效率。针对性能瓶颈, 提高硬件配置, 可以提高 Oracle 数据库的查询、更新的速度。本小节将为读者介绍以下优化服务器硬件的方法。

(1) 配置较大的内存。足够大的内存, 是提高 Oracle 数据库性能的方法之一。内存的速

度比磁盘 I/O 快得多, 可以通过增加系统的缓冲区容量, 使数据在内存停留的时间更长, 以减少磁盘 I/O。

(2) 配置高速磁盘系统, 以减少读盘的等待时间, 提高响应速度。

(3) 合理分布磁盘 I/O, 把磁盘 I/O 分散在多个设备上, 以减少资源竞争, 提高并行操作能力。

(4) 配置多处理器, Oracle 是多线程的数据库, 多处理器可同时执行多个线程。

17.4.2 优化 Oracle 的参数

通过优化 Oracle 的参数可以提高资源利用率, 从而达到提高 Oracle 服务器性能的目的。为了访问数据库中的数据, Oracle 数据库为所有用户提供一组后台进程, 并且有一些存储结构专门用来存储最近的有关对数据库访问的数据。这些存储区域可以通过减少对数据库文件的 I/O 次数来改善数据库性能。

数据库实例就是用来访问数据库文件集的一个存储结构以及后台进程的集合。它使一个单独的数据库可以被多个实例访问。决定实例的组成以及大小的参数存储在文件 `init.ora` 中。这个文件在实例启动时候需要装载, 也可以在运行中被装载。

通常需要设置的参数如下:

1. DB_BLOCK_BUFFERS

该参数决定了数据库缓冲区的大小, 这部分内存的作用主要是在内存中缓存从数据库中读取的数据块, 数据库缓冲区越大, 为用户已经在内存中的共享数据提供的内存就越大, 这样可以减少所需要的磁盘物理读写次数。

2. shared_pool_size

参数 `shared_pool_size` 的作用是缓存已经被解析过的 SQL 语句, 使其能被重用使用, 而不用再解析。SQL 语句的解析非常消耗 CPU 的资源, 如果一条 SQL 语句已经存在, 则进行的仅是软解析, 这将大大提高数据库的运行效率。当然, 这部分内存也并非越大越好, 如果分配的内存太大, Oracle 数据库为了维护共享结构, 将付出更大的管理开销。

这个参数的设置建议在 150~500MB 之间。如果系统内存为 1GB, 该值可设为 150~200MB; 如果为 2GB, 该值设为 250~300MB; 每增加 1GB 内存, 该值增加 100MB; 但该值最大不应超过 500MB。

3. Sort_area_size

该参数是当查询需要排序的时候, Oracle 将使用这部分内存做排序, 当内存不足时, 使用临时表空间做排序。这个参数是针对会话 (session) 设置的, 不是针对整个数据库。即如果应用有 170 个数据库连接, 假设这些连接都做排序操作, 则 Oracle 会分配 8×170 等于 1360MB 内存做排序, 而这些内存是在 Oracle 的 SGA 区之外分配的, 即如果 SGA 区分配了 1.6GB 内存, Oracle 还需要额外的 1.3GB 内存做排序。

建议该值设置不超过 3MB, 当物理内存为 1GB 时, 该值宜设为 1MB 或更低 (如 512KB);

2GB 时可设为 2MB 但不论物理内存多大，该值也不应超过 3MB。

4. sort_area_retained_size

这个参数的含义是当排序完成后至少为 session 继续保留的排序内存的最小值，该值最大可设为等于 sort_area_size。这样设置的好处是可以提高系统性能，因为下次再做排序操作时不需要再临时申请内存，缺点是如果 sort_area_size 设得过大并且 session 数很多时，将导致系统内存不足。建议该值设为 sort_area_size 的 10%~20%，或者不设置（默认为 0）。

5. Log_buffer

Log_buffer 是重做日志缓冲区，对数据库的任何修改都按顺序被记录在该缓冲，然后由进程将它写入磁盘。当用户提交后、有 1/3 重做日志缓冲区未被写入磁盘、有大于 1MB 重做日志缓冲区未被写入磁盘。建议不论物理内存多大，该值统一设为 1MB。

6. SESSION_CACHED_CURSOR

该参数指定要高速缓存的会话游标的数量。对同一 SQL 语句进行多次语法分析后，它的会话游标将被移到该会话的游标高速缓存中。这样可以缩短语法分析的时间，因为游标被高速缓存，无须被重新打开。设置该参数有助于提高系统的运行效率，建议无论在任何平台都应被设为 50。

7. re_page_sga

该参数表示将把所有 SGA 装载到内存中，以便使该实例迅速达到最佳性能状态。这将增加例程启动和用户登录的时间，但在内存充足的系统上能减少缺页故障的出现。建议在 2GB 以上（含 2G）内存的系统都将该值设为 true。

8. ML_LOCKS

该参数表示所有用户获取的锁的最大数量。对每个表执行 DML 操作均需要一个 DML 锁。例如，如果 3 个用户修改 2 个表，就要求该值为 6。该值过小可能会引起死锁问题。建议该参数不应该低于 600。

9. DB_FILE_MULTIBLOCK_READ_COUNT

该参数主要同全表扫描有关。当 Oracle 在请求大量连续数据块的时候，该参数控制块的读入速率。该参数能对系统性能产生较大的影响，建议把 DB_FILE_MULTIBLOCK_READ_COUNT 设为 8。

10. OPEN_CURSORS

指定一个会话一次可以打开的游标的最大数量，并且限制游标高速缓存的大小，以避免用户再次执行语句时重新进行语法分析。请将该值设置得足够高，这样才能防止应用程序耗尽打开的游标。此值建议设置为 250~300。

合理地配置这些参数可以提高 Oracle 服务器的性能。配置完参数以后，需要重新启动 Oracle 服务才会生效。



17.5 疑难解惑

疑问 1：是不是索引建立得越多越好？

合理的索引可以提高查询的速度，但不是索引越多越好。在执行插入语句的时候，Oracle 要为新插入的记录建立索引。所以，过多的索引会导致插入操作变慢。原则上是只有查询用的字段才建立索引。

疑问 2：为什么查询语句中的索引没有起作用？

在一些情况下，查询语句中使用了带有索引的字段，但索引并没有起作用。例如，在 WHERE 条件的 LIKE 关键字匹配的字符串以 “%” 开头，这种情况下索引不会起作用。又如，WHERE 条件中使用 OR 关键字连接查询条件，如果有 1 个字段没有使用索引，那么其他的索引也不会起作用。如果使用多列索引，但没有使用多列索引中的第 1 个字段，那么多列索引也不会起作用。

17.6 经典习题

- (1) 练习修改系统全局区。
- (2) 练习修改进程全局区。
- (3) 分析查询语句的执行计划。
- (4) 练习将很大的表分解成多个表，并观察分解表对性能的影响。
- (5) 练习使用中间表优化查询。
- (6) 练习优化 Oracle 服务器的配置参数。



第 18 章

◀ 设计新闻发布系统数据库 ▶

Oracle 数据库的使用非常广泛，很多的网站和管理系统使用 Oracle 数据库存储数据。本章主要讲述新闻发布系统的数据库设计过程。通过本章的学习，读者可以在新闻发布系统的设计过程中学会如何使用 Oracle 数据库。

- 了解新闻发布系统的概述
- 熟悉新闻发布系统的功能
- 掌握如何设计新闻发布系统的表
- 掌握如何设计新闻发布系统的索引
- 掌握如何设计新闻发布系统的视图
- 掌握如何设计新闻发布系统的触发器

18.1 系统概述

本章介绍的是一个小型新闻发布系统，管理员可以通过该系统发布新闻信息、管理新闻信息。一个典型的新闻发布系统网站至少应包含新闻信息管理、新闻信息显示和新闻信息查询 3 种功能。

新闻发布系统要实现的功能具体包括：新闻信息添加、新闻信息修改、新闻信息删除、显示全部新闻信息、按类别显示新闻信息、按关键字查询新闻信息、按关键字进行站内查询。

本站为一个简单的新闻信息发布系统，该系统具有以下特点。实用：系统实现了一个完整的信息查询过程。简单易用：为了使用户尽快掌握和使用整个系统，系统结构简单但功能齐全，简洁的页面设计使用户操作非常简便。代码规范：作为一个实例，文中的代码规范简洁、清晰易懂。

本系统主要用于实现发布新闻信息、管理用户、管理权限、管理评论等功能。这些信息的录入、查询、修改和删除等操作都是该系统重点解决的问题。

本系统主要功能包括以下几点：



- (1) 具有用户注册及个人信息管理功能。
- (2) 管理员可以发布新闻、删除新闻。
- (3) 用户注册后可以对新闻进行评论、发表留言。
- (4) 管理员可以管理留言和对用户进行管理。

18.2 系统功能

新闻发布系统分为 5 个管理部分，即用户管理、管理员管理、权限管理、新闻管理和评论管理。本系统的功能模块如图 18-1 所示。

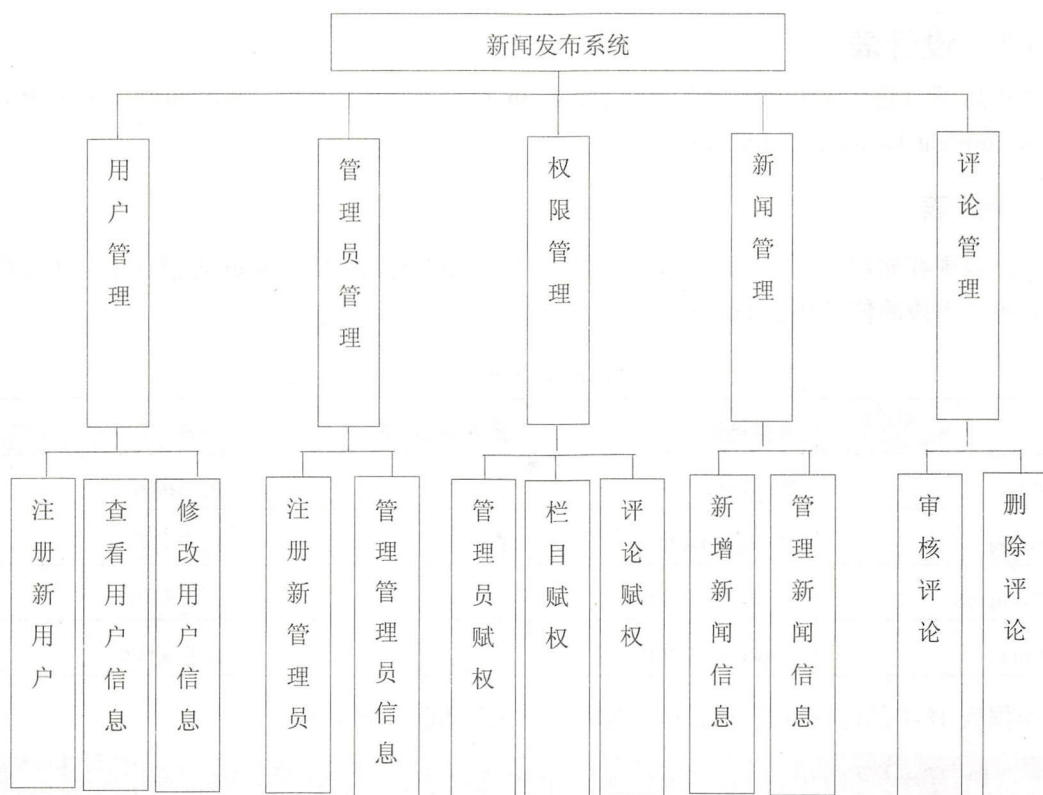


图 18-1 系统功能模块图

图 18-1 中模块的详细介绍如下。

- (1) 用户管理模块：实现新增用户，查看和修改用户信息功能。
- (2) 管理员管理模块：实现新增管理员，查看、修改和删除管理员信息功能。
- (3) 权限管理模块：实现对管理员、对管理的模块和管理的评论赋权功能。
- (4) 新闻管理模块：实现有相关权限的管理员对新闻的增加、查看、修改和删除功能。
- (5) 评论管理模块：实现有相关权限的管理员对评论的审核和删除功能。



通过本节的介绍，读者对这个新闻发布系统的主要功能有了一定的了解，下一节会向读者介绍本系统所需要的数据库和表。

18.3 数据库设计和实现

数据库设计是开发管理系统的最重要的一个步骤。如果数据库设计得不够合理，将会为后续的开发工作带来很大的麻烦。本节为读者介绍新闻发布系统的数据库开发过程。

数据库设计时要确定设计哪些表、表中包含哪些字段、字段的数据类型和长度。通过本章的学习，读者可以对 Oracle 数据库的知识有一个全面的了解。

18.3.1 设计表

数据库下总共存放 9 张表，分别是 user、admin、roles、news、category、comment、admin_Roles、news_Comment 和 users_Comment。

1. user 表

user 表中存储用户 ID、用户名、密码和用户 Email 地址，所以 user 表设计了 4 个字段。user 表每个字段的信息如表 18-1 所示。

表 18-1 user 表的内容

列名	数据类型	允许 NULL 值	说明
userID	NUMBER (9)	否	用户编号
userName	VARCHAR2(20)	否	用户名称
userPassword	VARCHAR2(20)	否	用户密码
userEmail	VARCHAR2(20)	否	用户 Email

根据表 18-1 的内容创建 user 表。创建 user 表的 SQL 语句如下：

```
CREATE TABLE user(  
  userID NUMBER (9) PRIMARY KEY UNIQUE NOT NULL,  
  userName VARCHAR2(20) NOT NULL,  
  userPassword VARCHAR2(20) NOT NULL,  
  userEmail VARCHAR2(20) NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 user 表的基本结构。



2. admin 表

管理员信息表（admin）主要用来存放用户账号信息，如表 18-2 所示。

表 18-2 admin 表的内容

列名	数据类型	允许 NULL 值	说明
adminID	NUMBER (9)	否	管理员编号
adminName	VARCHAR2(20)	否	管理员名称
adminPassword	VARCHAR2(20)	否	管理员密码

根据表 18-2 的内容创建 admin 表。创建 admin 表的 SQL 语句如下：

```
CREATE TABLE admin(  
adminID NUMBER (9) PRIMARY KEY UNIQUE NOT NULL,  
adminName VARCHAR2 (20) NOT NULL,  
adminPassword VARCHAR2 (20) NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 admin 表的基本结构。

3. roles 表

权限信息表（roles）主要用来存放权限信息，如表 18-3 所示。

表 18-3 roles 表的内容

列名	数据类型	允许 NULL 值	说明
roleID	NUMBER (9)	否	权限编号
roleName	VARCHAR2(20)	否	权限名称

根据表 18-3 的内容创建 roles 表。创建 roles 表的 SQL 语句如下：

```
CREATE TABLE roles(  
roleID NUMBER (9) PRIMARY KEY UNIQUE NOT NULL,  
roleName VARCHAR2 (20) NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 roles 表的基本结构。

4. news 表

新闻信息表（news）主要用来存放新闻信息，如表 18-4 所示。



表 18-4 news 表的内容

列名	数据类型	允许 NULL 值	说明
newsID	NUMBER (9)	否	新闻编号
newsTitle	VARCHAR2(50)	否	新闻标题
newsContent	VARCHAR2(500)	否	新闻内容
newsDate	DATE	是	发布时间
newsDesc	VARCHAR2(50)	否	新闻描述
newsImagePath	VARCHAR2(50)	是	新闻图片路径
newsRate	NUMBER(9)	否	新闻级别
newsIsCheck	VARCHAR2(2)	否	新闻是否检验
newsIsTop	VARCHAR2(2)	否	新闻是否置顶

根据表 18-4 的内容创建 news 表。创建 news 表的 SQL 语句如下：

```
CREATE TABLE news (  
newsID NUMBER (9) PRIMARY KEY UNIQUE NOT NULL,  
newsTitle VARCHAR2(50) NOT NULL,  
newsContent VARCHAR2(500) NOT NULL,  
newsDate DATE,  
newsDesc VARCHAR2(50) NOT NULL,  
newsImagePath VARCHAR2(50),  
newsRate NUMBER (9) NOT NULL,  
newsIsCheck VARCHAR2(2) NOT NULL,  
newsIsTop VARCHAR2(2) NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 news 表的基本结构。

5. category 表

栏目信息表（category）主要用来存放新闻栏目信息，如表 18-5 所示。

表 18-5 category 表的内容

列名	数据类型	允许 NULL 值	说明
categoryID	NUMBER (9)	否	栏目编号
categoryName	VARCHAR2(50)	否	栏目名称
categoryDesc	VARCHAR2(50)	否	栏目描述



根据表 18-5 的内容创建 category 表。创建 category 表的 SQL 语句如下：

```
CREATE TABLE category (  
categoryID NUMBER(9) PRIMARY KEY UNIQUE NOT NULL,  
categoryName VARCHAR2(50) NOT NULL,  
categoryDesc VARCHAR2(50) NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 category 表的基本结构。

6. comment 表

评论信息表（comment）主要用来存放新闻评论信息，如表 18-6 所示。

表 18-6 comment 表的内容

列名	数据类型	允许 NULL 值	说明
commentID	NUMBER(9)	否	评论信息编号
commentTitle	VARCHAR2(50)	否	评论标题
commentContent	VARCHAR2(500)	否	评论内容
commentDate	DATE	否	评论日期

根据表 18-6 的内容创建 comment 表。创建 comment 表的 SQL 语句如下：

```
CREATE TABLE comment (  
commentID NUMBER(9) PRIMARY KEY UNIQUE NOT NULL,  
commentTitle VARCHAR2(50) NOT NULL,  
commentContent VARCHAR2(500) NOT NULL,  
commentDate DATE NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 comment 表的基本结构。

7. admin_Roles 表

管理员_权限表（admin_Roles）主要用来存放管理员和权限的关系，如表 18-7 所示。

表 18-7 admin_Roles 表的内容

列名	数据类型	允许 NULL 值	说明
aRID	NUMBER(9)	否	管理员_权限编号
adminID	NUMBER(9)	否	管理员编号
roleID	NUMBER(9)	否	权限编号



根据表 18-7 的内容创建 admin_Roles 表。创建 admin_Roles 表的 SQL 语句如下：

```
CREATE TABLE admin_Roles (  
  aRID NUMBER (9) PRIMARY KEY UNIQUE NOT NULL,  
  adminID NUMBER (9) NOT NULL,  
  roleID NUMBER (9) NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 admin_Roles 表的基本结构。

8. news_Comment 表

新闻_评论表（news_Comment）主要用来存放新闻和评论的关系，如表 18-8 所示。

表 18-8 新闻_评论表

列名	数据类型	允许 NULL 值	说明
nCommentID	NUMBER (9)	否	新闻_评论编号
newsID	NUMBER (9)	否	新闻编号
commentID	NUMBER (9)	否	评论编号

根据表 18-8 的内容创建 news_Comment 表。创建 news_Comment 表的 SQL 语句如下：

```
CREATE TABLE news_Comment (  
  nCommentID NUMBER (9) PRIMARY KEY UNIQUE NOT NULL,  
  newsID NUMBER (9) NOT NULL,  
  commentID NUMBER (9) NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 news_Comment 表的基本结构。

9. users_Comment 表

用户_评论表（users_Comment）主要用来存放用户和评论的关系，如表 18-9 所示。

表 18-9 用户_评论表

列名	数据类型	允许 NULL 值	说明
uCID	NUMBER (9)	否	用户_评论编号
userID	NUMBER (9)	否	用户编号
commentID	NUMBER (9)	否	评论编号

根据表 18-9 的内容创建 users_Comment 表。创建 users_Comment 表的 SQL 语句如下：

```
CREATE TABLE news_Comment (  
  uCID NUMBER (9) PRIMARY KEY UNIQUE NOT NULL,  
  userID NUMBER (9) NOT NULL,  
  commentID NUMBER (9) NOT NULL  
);
```



```
uCID NUMBER (9) PRIMARY KEY UNIQUE NOT NULL,  
userID NUMBER (9) NOT NULL,  
commentID NUMBER (9) NOT NULL  
);
```

创建完成后, 可以使用 DESC 语句查看 users_Comment 表的基本结构。

18.3.2 设计索引

索引是创建在表上的, 是对数据库中一列或者多列的值进行排序的一种结构。索引可以提高查询的速度。新闻发布系统需要查询新闻的信息, 这就需要在某些特定字段上建立索引, 以便提高查询速度。

1. 在 news 表上建立索引

新闻发布系统中需要按照 newsTitle 字段、newsDate 字段和 newsRate 字段查询新闻信息。本书在前面的章节中介绍了几种创建索引的方法。本小节将使用 CREATE INDEX 语句和 ALTER TABLE 语句创建索引。

下面使用 CREATE INDEX 语句在 newsTitle 字段上创建名为 index_new_title 的索引。SQL 语句如下:

```
CREATE INDEX index_new_title ON news(newsTitle);
```

然后, 再使用 CREATE INDEX 语句在 newsDate 字段上创建名为 index_new_date 的索引。SQL 语句如下:

```
CREATE INDEX index_new_date ON news(newsDate);
```

最后, 再使用 ALTER TABLE 语句在 newsRate 字段上创建名为 index_new_rate 的索引。SQL 语句如下:

```
ALTER TABLE news ADD INDEX index_new_rate (newsRate);
```

2. 在 category 表上建立索引

新闻发布系统中需要通过栏目名称查询该栏目下的新闻, 因此需要在这个字段上创建索引。创建索引的语句如下:

```
CREATE INDEX index_category_name ON category (categoryName);
```

代码执行完成后, 读者可以使用 SHOW CREATE TABLE 语句查看 category 表的详细信息。

3. 在 comment 表上建立索引

新闻发布系统需要通过 commentTitle 字段和 commentDate 字段查询评论内容, 因此可以在这两个字段上创建索引。创建索引的语句如下:



```
CREATE INDEX index_comment_title ON comment (commentTitle);  
CREATE INDEX index_comment_date ON comment (commentDate);
```

代码执行完成后，读者可以通过 SHOW CREATE TABLE 语句查看 comment 表的结构。

18.3.3 设计视图

视图是由数据库中一个表或者多个表导出的虚拟表，其作用是方便用户对数据的操作。在这个新闻发布系统中，也设计了一个视图改善查询操作。

在新闻发布系统中，如果直接查询 news_Comment 表，显示信息时会显示新闻编号和评论编号。这种显示不直观，为了以后查询方便，可以建立一个视图 news_view。这个视图显示评论编号、新闻编号、新闻级别、新闻标题、新闻内容和新闻发布时间。创建视图 news_view 的 SQL 代码如下：

```
CREATE VIEW news_view  
AS  
SELECT  
c.commentID,n.newsID,n.newsRate,n.newsTitle,n.newsContent,n.newsDate  
FROM news_Comment c,news n  
WHERE news_Comment.newsID=news.newsID;
```

上述 SQL 语句中给每个表都取了别名，news_Comment 表的别名为 c，news 表的别名为 n。这个视图从这两个表中取出相应的字段。视图创建完成后，可以使用 SHOW CREATE VIEW 语句查看 news_view 视图的详细信息。

18.3.4 设计触发器

触发器是由 INSERT、UPDATE 和 DELETE 等事件来触发某种特定的操作。满足触发器的触发条件时，数据库系统就会执行触发器中定义的程序语句。这样做可以保证某些操作之间的一致性。为了使新闻发布系统的数据更新更加快速和合理，可以在数据库中设计几个触发器。

1. 设计 UPDATE 触发器

在设计表时，news 表和 news_Comment 表的 newsID 字段的值是一样的。如果 news 表中的 newsID 字段的值更新了，那么 news_Comment 表中的 newsID 字段的值也必须同时更新。这可以通过一个 UPDATE 触发器来实现。创建 UPDATE 触发器 UPDATE_NEWSID 的 SQL 代码如下：

```
CREATE TRIGGER UPDATE_NEWSID  
AFTER UPDATE  
ON news  
FOR EACH ROW  
BEGIN
```

```
UPDATE news_Comment SET newsID=NEW.newsID;  
END
```

其中，NEW.newsID 表示 news 表中更新的记录的 newsID 值。

2. 设计 DELETE 触发器

如果从 user 表中删除一个用户的信息，那么这个用户在 users_Comment 表中的信息也必须同时删除。这也可以通过触发器来实现。在 user 表上创建 DELETE_USER 触发器，只要执行 DELETE 操作，那么就删除 users_Comment 表中相应的记录。创建 DELETE_USER 触发器的 SQL 语句如下：

```
CREATE TRIGGER DELETE_USER  
AFTER DELETE  
ON user  
FOR EACH ROW  
BEGIN  
    DELETE FROM users_Comment WHERE userID=OLD.userID;  
END
```

其中，OLD.userID 表示新删除的记录的 userID 值。

18.4 小结

本章介绍了设计新闻发布系统数据库的方法。本章的重点是数据库的设计部分。因为本书主要介绍 Oracle 数据库的使用，所以数据库设计部分是本章的主要内容。在数据库设计方面，不仅涉及了表和字段的设计，还涉及索引、视图和触发器等内容。其中，为了提高表的查询速度，有意识地在表中增加了冗余字段，这是数据库的性能优化的内容。希望通过本章的学习，读者可以对 Oracle 数据库有一个全新的认识。

第 19 章

◀ 设计论坛管理系统数据库 ▶

随着论坛的出现，人们的交流有了新的变化。在论坛里，人们之间的交流打破了空间、时间的限制。在论坛系统中，用户可以注册成为论坛会员，取得发表言论的资格，也需要论坛信息管理工作系统化、规范化、自动化。通过这样的系统，可以做到信息的规范管理、科学统计和快速地发表言论。为了实现论坛系统规范和稳健运行，这就需要数据库的设计非常合理。本章主要讲述论坛管理系统数据库的设计方法。

- 了解论坛系统的概述
- 熟悉论坛系统的功能
- 掌握如何设计论坛系统的方案图表
- 掌握如何设计论坛发布系统的表
- 掌握如何设计论坛发布系统的索引
- 掌握如何设计论坛发布系统的视图
- 掌握如何设计论坛发布系统的触发器

19.1 系统概述

论坛又名 BBS，全称为 Bulletin Board System（电子公告板）或者 Bulletin Board Service（公告板服务）。它是 Internet 上的一种电子信息服务系统。它提供一块公共电子白板，每个用户都可以在上面书写，可发布信息或提出看法。

论坛是一种交互性强、内容丰富而及时的电子信息服务系统。用户在 BBS 站点上可以获得各种信息服务、发布信息、进行讨论、聊天等。像日常生活中的黑板报一样，论坛按不同的主题分为许多版块，版面的设立依据是大多数用户的要求和喜好，用户可以阅读别人关于某个主题的看法，也可以将自己的想法毫无保留地发表到论坛中。随着计算机网络技术的不断发展，BBS 论坛的功能越来越强大，目前 BBS 的主要功能有以下几点：

- （1）供用户自我选择阅读若干感兴趣的专业组和讨论组内的信息。
- （2）可随意检查是否有新消息发布并选择阅读。



- (3) 用户可在站点内发布消息或文章供他人查阅。
- (4) 用户可就站点内其他人的消息或文章进行评论。
- (5) 同一站点内的用户互通电子邮件，设定好友名单。

现实生活中的交流存在时间和空间上的局限性，交流人群范围的狭小，以及间断的交流，不能保证信息的准确性和可取性。因此，用户需要通过网上论坛也就是 BBS 的交流扩大交流面，同时可以从多方面获得自己的及时需求。同时，信息时代迫切要求信息传播速度加快，局部范围的信息交流只会减缓前进的步伐。

BBS 系统的开发能为分散于五湖四海的人提供一个共同交流、学习、倾吐心声的平台，实现来自不同地方用户的极强的信息互动性，用户在获得自己所需要的信息的同时，也可以广交朋友拓展自己的视野和扩大自己的社交面。

论坛系统的基本功能包括用户信息的录入、查询、修改和删除，用户留言及头像的前台显示功能。其中，还包括管理员的登录信息。

19.2 系统功能

论坛管理系统的重要功能是管理论坛帖子的基本信息。通过本管理系统，可以提高论坛管理员的工作效率。本节将详细介绍本系统的功能。

论坛系统主要分为 5 个管理部分，包括用户管理、管理员管理、版块管理、主帖管理和回帖管理。本系统的功能模块图如图 19-1 所示。

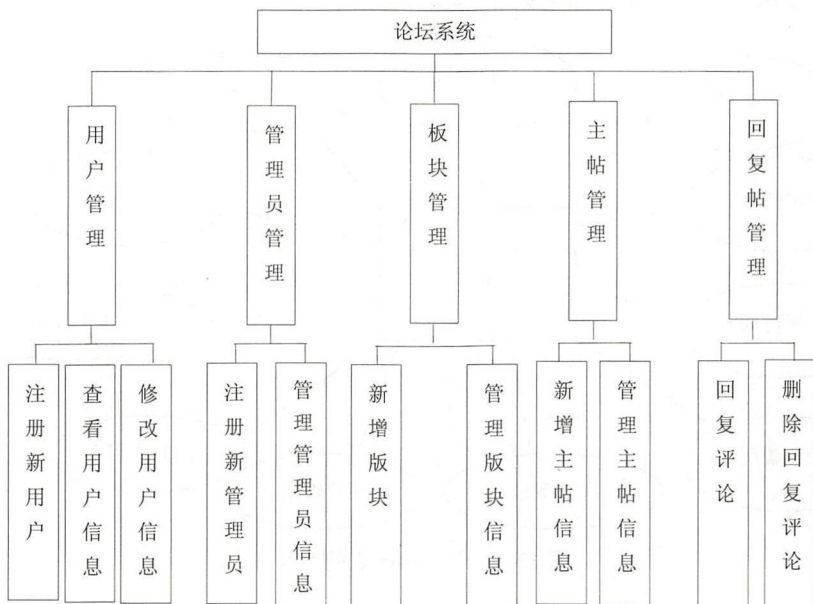


图 19-1 系统功能模块图



图 19-1 中模块的详细介绍如下。

- (1) 用户管理模块：实现新增用户，查看和修改用户信息功能。
- (2) 管理员管理模块：实现新增管理员，查看、修改和删除管理员信息功能。
- (3) 版块管理模块：实现新增版块，查看、修改和删除版块信息功能。
- (4) 主帖管理模块：实现对主帖的增加、查看、修改和删除功能。
- (5) 回复帖管理模块：实现有相关权限的管理员对回复帖的审核和删除功能。

通过本节介绍，读者对这个论坛系统的主要功能有了一定的了解，下一节会向读者介绍本系统所需要的数据库和表。

19.3 数据库设计和实现

数据库设计时要确定设计哪些表、表中包含哪些字段、字段的数据类型和长度。本章节主要讲述论坛数据库的设计和实现过程。

19.3.1 设计方案图表

在设计表之前，用户可以先设计出方案图表。

1. 用户表的 E-R 图

用户管理的表为 user，E-R 图如图 19-2 所示。

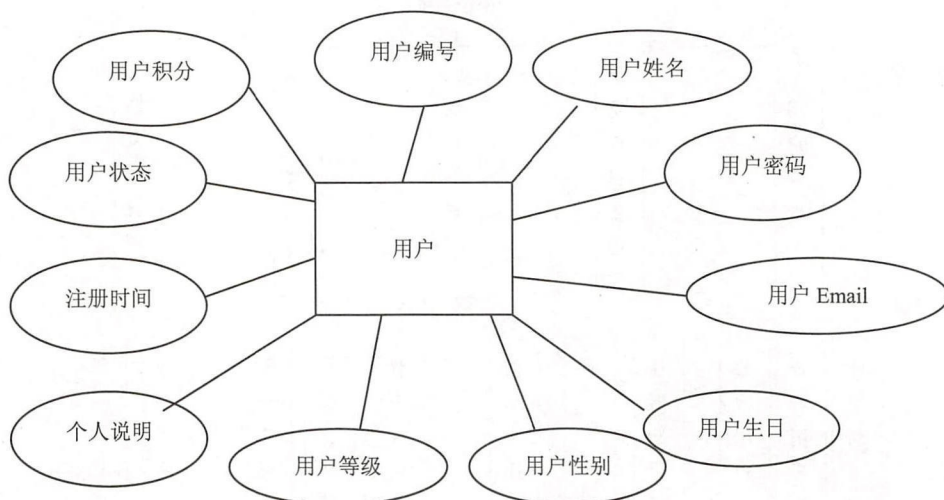


图 22-2 用户 user 表的 E-R 图



2. 管理员表的 E-R 图

管理员管理的表为 admin，E-R 图如图 19-3 所示。

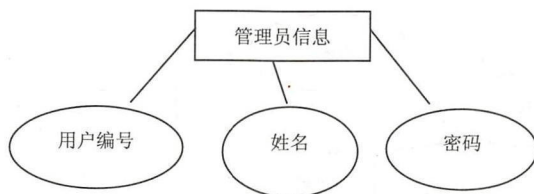


图 19-3 管理员 admin 表的 E-R 图

3. 版块表的 E-R 图

版块管理的表为 section，E-R 图如图 19-4 所示。

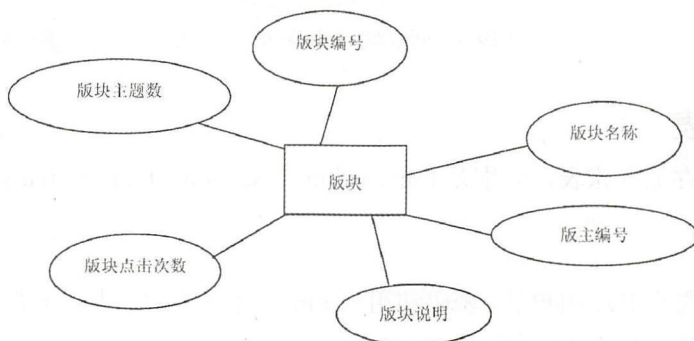


图 19-4 版块 section 表的 E-R 图

4. 主帖表的 E-R 图

主帖管理的表为 topic，E-R 图如图 19-5 所示。

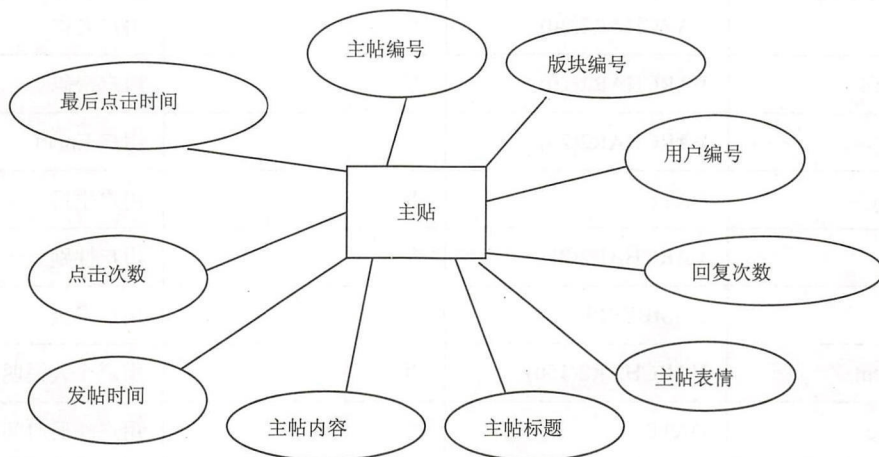


图 19-5 主贴 topic 表的 E-R 图



5. 回复帖表的 E-R 图

回复帖管理的表为 reply，E-R 图如图 19-6 所示。

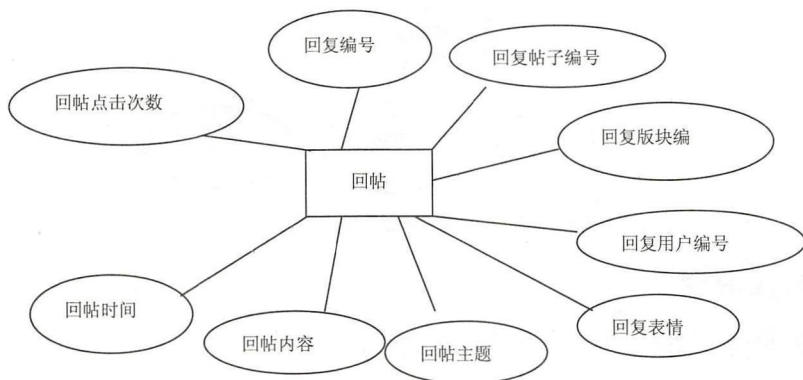


图 19-6 回复帖 reply 表的 E-R 图

19.3.2 设计表

数据库下总共存放 5 张表，分别是 user、admin、section、topic 和 reply。

1. user 表

user 表中存储用户 ID、用户名、密码和用户 Email 地址等，所以 user 表设计了 10 个字段。user 表每个字段的信息如表 19-1 所示。

表 19-1 user 表的内容

列名	数据类型	允许 NULL 值	说明
uID	NUMBER(9)	否	用户编号
userName	VARCHAR2(20)	否	用户名称
userPassword	VARCHAR2(20)	否	用户密码
userEmail	VARCHAR2(20)	否	用户 Email
userBirthday	DATE	否	用户生日
userSex	VARCHAR2(2)	否	用户性别
userClass	NUMBER(9)	否	用户等级
userStatement	VARCHAR2(150)	否	用户个人说明
userRegDate	DATE	否	用户注册时间
userPoint	NUMBER(9)	否	用户积分



根据表 19-1 的内容创建 user 表。创建 user 表的 SQL 语句如下：

```
CREATE TABLE user(  
  uID NUMBER(9) PRIMARY KEY UNIQUE NOT NULL,  
  userName VARCHAR2(20) NOT NULL,  
  userPassword VARCHAR2(20) NOT NULL,  
  userEmail VARCHAR2(20) NOT NULL,  
  userBirthday DATE NOT NULL,  
  userSex VARCHAR(2) NOT NULL,  
  userClass NUMBER(9) NOT NULL,  
  userStatement VARCHAR2(150) NOT NULL,  
  userRegDate DATE NOT NULL,  
  userPoint NUMBER(9) NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 user 表的基本结构。

2. admin 表

管理员信息表（admin）主要用来存放用户账号信息，如表 19-2 所示。

表 19-2 admin 表的内容

列名	数据类型	允许 NULL 值	说明
adminID	NUMBER(9)	否	管理员编号
adminName	VARCHAR2(20)	否	管理员名称
adminPassword	VARCHAR2(20)	否	管理员密码

根据表 19-2 的内容创建 admin 表。创建 admin 表的 SQL 语句如下：

```
CREATE TABLE admin(  
  adminID NUMBER(9) PRIMARY KEY UNIQUE NOT NULL,  
  adminName VARCHAR2(20) NOT NULL,  
  adminPassword VARCHAR2(20) NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 admin 表的基本结构。

3. section 表

版块信息表（section）主要用来存放版块信息，如表 19-3 所示。



表 19-3 section 表的内容

列名	数据类型	允许 NULL 值	说明
sID	NUMBER(9)	否	版块编号
sName	VARCHAR2(20)	否	版块名称
sMasterID	NUMBER(9)	否	版主编号
sStatement	VARCHAR2(20)	否	版块说明
sClickCount	NUMBER(9)	否	版块点击次数
sTopicCount	NUMBER(9)	否	版块主题数

根据表 19-3 的内容创建 section 表。创建 section 表的 SQL 语句如下：

```
CREATE TABLE section (  
  sID NUMBER(9) PRIMARY KEY UNIQUE NOT NULL,  
  sName VARCHAR2(20) NOT NULL,  
  sMasterID NUMBER(9) NOT NULL,  
  sStatement VARCHAR2(20) NOT NULL,  
  sClickCount NUMBER(9) NOT NULL,  
  sTopicCount NUMBER(9) NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 section 表的基本结构。

4. topic 表

主帖信息表（topic）主要用来存放主帖信息，如表 19-4 所示。

表 19-4 topic 表的内容

列名	数据类型	允许 NULL 值	说明
tID	NUMBER(9)	否	主帖编号
tSID	NUMBER(9)	否	主帖版块编号
tuid	NUMBER(9)	否	主帖用户编号
tReplyCount	NUMBER(9)	否	主帖回复次数
tEmotion	VARCHAR2(20)	否	主帖表情
tTopic	VARCHAR2(20)	否	主帖标题



(续表)

列名	数据类型	允许 NULL 值	说明
tContents	VARCHAR2(500)	否	主帖内容
tTime	DATE	否	发帖时间
tClickCount	NUMBER(9)	否	主帖点击次数
tLastClickT	DATE	否	主帖最后点击时间

根据表 19-4 的内容创建 topic 表。创建 topic 表的 SQL 语句如下：

```
CREATE TABLE topic (  
  tID NUMBER(9) PRIMARY KEY UNIQUE NOT NULL,  
  tSID NUMBER(9) NOT NULL,  
  tuid NUMBER(9) NOT NULL,  
  tReplyCount NUMBER(9) NOT NULL,  
  tEmotion VARCHAR2(20) NOT NULL,  
  tTopic VARCHAR2(20) NOT NULL,  
  tContents VARCHAR2(500) NOT NULL,  
  tTime DATE NOT NULL,  
  tClickCount NUMBER(9) NOT NULL,  
  tLastClickT DATE NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 topic 表的基本结构。

5. reply 表

回复帖信息表（reply）主要用来存放回复帖的信息，如表 19-5 所示。

表 19-5 reply 表的内容

列名	数据类型	允许 NULL 值	说明
rID	NUMBER(9)	否	回复编号
tID	NUMBER(9)	否	回复帖子编号
uID	NUMBER(9)	否	回复用户编号
rEmotion	VARCHAR2(4)	否	回帖表情
rTopic	VARCHAR2(20)	否	回帖主题



(续表)

列名	数据类型	允许 NULL 值	说明
rContents	VARCHAR2(200)	否	回帖内容
rTime	DATE	否	回帖时间
rClickCount	NUMBER(9)	否	回帖点击次数

根据表 19-5 的内容创建 reply 表。创建 reply 表的 SQL 语句如下：

```
CREATE TABLE reply (  
  rID NUMBER(9) PRIMARY KEY UNIQUE NOT NULL,  
  tID NUMBER(9) NOT NULL,  
  uID NUMBER(9) NOT NULL,  
  rEmotion VARCHAR2 (4) NOT NULL,  
  rTopic VARCHAR2 (20) NOT NULL,  
  rContents VARCHAR2 (200) NOT NULL,  
  rTime DATE NOT NULL,  
  rClickCount NUMBER(9) NOT NULL  
);
```

创建完成后，可以使用 DESC 语句查看 reply 表的基本结构。

19.3.3 设计索引

索引是创建在表上的，是对数据库中一列或者多列的值进行排序的一种结构。索引可以提高查询的速度。论坛系统需要查询论坛的信息，这就需要在某些特定字段上建立索引，以便提高查询速度。

1. 在 topic 表上建立索引

论坛系统中需要按照 tTopic 字段、tTime 字段和 tContents 字段查询主帖信息。本书在前面的章节中介绍了几种创建索引的方法。本小节将使用 CREATE INDEX 语句和 ALTER TABLE 语句创建索引。

下面使用 CREATE INDEX 语句在 tTopic 字段上创建名为 index_topic_title 的索引。SQL 语句如下：

```
CREATE INDEX index_topic_title ON topic(tTopic);
```

然后，再使用 CREATE INDEX 语句在 tTime 字段上创建名为 index_topic_time 的索引。SQL 语句如下：



```
CREATE INDEX index_topic_time ON topic(tTime);
```

最后，再使用 ALTER TABLE 语句在 tContents 字段上创建名为 index_topic_contents 的索引。SQL 语句如下：

```
ALTER TABLE topic ADD INDEX index_topic_contents (tContents);
```

2. 在 section 表上建立索引

论坛系统中需要通过版块名称查询该版块下的帖子信息，因此需要在这个字段上创建索引。创建索引的语句如下：

```
CREATE INDEX index_section_name ON section (sName);
```

代码执行完成后，读者可以使用 SHOW CREATE TABLE 语句查看 section 表的详细信息。

3. 在 reply 表上建立索引

论坛系统需要通过 rTime 字段、rTopic 字段和 tID 字段查询回复帖子的内容，因此可以在这 3 个字段上创建索引。创建索引的语句如下：

```
CREATE INDEX index_reply_rtime ON reply (rTime);  
CREATE INDEX index_reply_rtopic ON reply (rTopic);  
CREATE INDEX index_reply_tid ON reply (tID);
```

代码执行完成后，读者可以通过 SHOW CREATE TABLE 语句查看 reply 表的结构。

19.3.4 设计视图

在论坛系统中，如果直接查询 section 表，显示信息时会显示版块编号和版块名称等信息。这种显示不直观，为了以后查询方便，可以建立一个视图 topic_view。这个视图显示版块的编号、版块的名称、同一版块下主帖的标题、主帖的内容和主帖的发布时间。创建视图 topic_view 的 SQL 代码如下：

```
CREATE VIEW topic_view  
AS SELECT s.ID,s.Name,t.tTopic,t.tContents,t.tTime  
FROM section s,topic t  
WHERE section.sID=topic.sID;
```

上述 SQL 语句中给每个表都取了别名，section 表的别名为 s；topic 表的别名为 t，这个视图从这两个表中取出相应的字段。视图创建完成后，可以使用 SHOW CREATE VIEW 语句查看 topic_view 视图的详细信息。

19.3.5 设计触发器

触发器是由 INSERT、UPDATE 和 DELETE 等事件来触发某种特定的操作。满足触发器



的触发条件时，数据库系统就会执行触发器中定义的程序语句。这样做可以保证某些操作之间的一致性。为了使论坛系统的数据更新更加快速和合理，可以在数据库中设计几个触发器。

1. 设计 INSERT 触发器

如果向 section 表插入记录，说明版块的主题数目要相应地增加。这可以通过触发器来完成。在 section 表上创建名为 SECTION_COUNT 的触发器，其 SQL 语句如下：

```
CREATE TRIGGER SECTION_COUNT
AFTER INSERT
ON section
FOR EACH ROW
BEGIN
    UPDATE section SET sTopicCount= sTopicCount+1
    WHERE sID=NEW.sID;
END
```

其中，NEW.sID 表示 section 表中增加的记录 sID 值。

2. 设计 UPDATE 触发器

在设计表时，user 表和 reply 表的 uID 字段的值是一样的。如果 user 表中的 uID 字段的值更新了，那么 reply 表中的 uID 字段的值也必须同时更新。这可以通过一个 UPDATE 触发器来实现。创建 UPDATE 触发器 UPDATE_USERID 的 SQL 代码如下：

```
CREATE TRIGGER UPDATE_USERID
AFTER UPDATE
ON user
FOR EACH ROW
BEGIN
    UPDATE reply SET uID=NEW.uID;
END
```

其中，NEW.uID 表示 user 表中更新的记录的 uID 值。

3. 设计 DELETE 触发器

如果从 user 表中删除一个用户的信息，那么这个用户在 topic 表中的信息也必须同时删除。这也可以通过触发器来实现。在 user 表上创建 DELETE_USER 触发器，只要执行 DELETE 操作，那么就删除 topic 表中相应的记录。创建 DELETE_USER 触发器的 SQL 语句如下：

```
CREATE TRIGGER DELETE_USER
```

```
AFTER DELETE
ON user
FOR EACH ROW
BEGIN
    DELETE FROM topic WHERE uID=OLD.uID;
END
```

其中，OLD.uID 表示新删除的记录 uID 值。

19.4 小结

本章介绍了设计论坛系统数据库的方法。本章的重点是数据库的设计部分。在数据库设计方面，不仅涉及表和字段的设计，还涉及索引、视图和触发器等内容。特别是新增加了设计方案图表，通过图表的设计，用户可以清晰地看到各个表的设计字段和各个字段的关系。希望通过本章的学习，读者可以对论坛数据库的设计有一个清晰的思路。

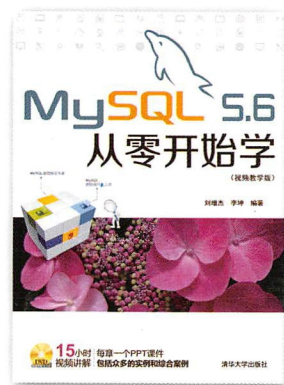


图书推荐

本书全面讲解MySQL 5.6的安装与配置、数据库的创建、数据表的创建、数据类型和运算符、MySQL函数、查询数据、数据表的操作、索引、存储过程和函数、视图、触发器、用户管理、数据备份与还原、日志、性能优化、MySQL Replication、MySQL Workbench、MySQL Cluster集群技术等内容。

本书注重实战操作，提供大量的实例和案例，帮助读者循序渐进地掌握MySQL中的各项技术。

本书光盘包括所有实例源码，课件，以及近20小时培训班形式的视频教学录像，方便读者学习并深入掌握MySQL数据库操作方法和技巧。



清华大学出版社数字出版网站

WQBook 书文局泉

www.wqbook.com

